Logique

méthodes pour l'intelligence artificielle

Paul Gochet
Pascal Gribomont
André Thayse

Hermes Science
publications

REMERCIEMENTS

Les auteurs remercient Patrick Blackburn, Maarten de Rijke, Pascal Fontaine, Marie-Hélène Foubert, Eric Gillet, Rajeev Goré, Thierry Lucas et Ide Venema qui ont accepté de lire et de commenter des parties de cet ouvrage.

© HERMES Science Publications, Paris, 2000

HERMES Science Publications
8, quai du Marché-Neuf
75004 Paris
Serveur web : http://www.hermes-science.com

ISBN 2-7462-0131-3

Catalogage Electre-Bibliographie
Gochet, Paul*Gribomont, Pascal*Thayse, André
Logique 3 – Méthodes pour l'intelligence artificielle
Paris, Hermès Science Publications, 2000
ISBN 2-7462-0131-3
RAMEAU : logique
intelligence artificielle
DEWEY : 006.1 : Méthodes informatiques spéciales.
Intelligence artificielle

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5, d'une part, que les "copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective" et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, "toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite" (article L. 122-4). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

Paul Gochet
Pascal Gribomont
André Thayse

Logique

Volume 3

méthodes pour l'intelligence artificielle

Hermes Science publications

EXTRAIT DU CATALOGUE GÉNÉRAL

Apprentissage automatique, sous la direction de Marc SEBBAN, Gilles VENTURINI. 1999.

Information, complexité et hasard – *2e édition revue*, Jean-Paul DELAHAYE, 1999.

Logique – *volume 1. Méthodes pour l'informatique fondamentale*, Paul GOCHET, Pascal GRIBOMONT, 1990, 1991, 1998.

Apprentissage – *des principes naturels aux méthodes artificielles*, sous la direction de Gilbert RITSCHARD, André BERCHTOLD, François DUC, Djamel A. ZIGHED, 1998.

Génie logiciel et psychologie de la programmation, Françoise DÉTIENNE, 1998.

Sémantiques, sous la direction de Michel CHAMBREUIL, 1998.

Auto-organisation et comportement, coordonnateurs Guy THERAULAZ, François SPITZ, 1997.

Eléments de logique floue, Louis GACÔGNE, 1997.

Les agents intelligents – *essai sur la rationalité des calculs*, Jean SALLANTIN, 1997.

La représentation des connaissances, Daniel KAYSER, 1997.

La polysémie – *construction dynamique du sens*, Bernard VICTORRI, Catherine FUCHS, 1996.

Langage et cognition – *l'approche connexionniste*, Bernard LAKS, 1996.

Eléments de logique formelle, Gérard CHAZAL, 1996.

Les systèmes de connaissances, Jean-Louis ERMINE, 1996.

Logique et complexité, Richard LASSAIGNE, Michel DE ROUGEMONT, 1996.

Méthodes logiques pour les sciences cognitives, coordonnateurs Jacques DUBUCS, François LEPAGE, 1995.

Logique – *volume 2. Méthodes formelles pour l'étude des programmes*, Paul GOCHET, Pascal GRIBOMONT, 1994.

Table des matières

Avant-propos 17

Chapitre 1. Introduction historique et philosophique à la logique modale 19

1.1. L'étude des modalités relève-t-elle de la logique ? 19
1.2. Le carré modal 20
1.3. Le syllogisme modal 21
1.4. Les futurs contingents 22
1.5. L'école de Mégare 23
1.6. Les modalités temporelles au Moyen Age 24
1.7. La logique déontique au Moyen Age 26
1.8. Les logiques multimodales 29
1.9. De l'implication 29
1.10. L'implication stricte 30
1.11. Polysémie de l'implication stricte 31
1.12. La correction des systèmes de Lewis 32
1.13. L'avènement de la logique modale quantifiée 33
1.14. Carnap, précurseur de la sémantique des mondes possibles 34
1.15. De la nécessité selon Carnap à la nécessité selon Kripke 35
1.16. Domaines variables et relation d'accessibilité 36
1.17. De l'implication stricte à l'implication relevante 38
1.18. La diversification de la logique modale 40

Chapitre 2. Logique modale des propositions 41

2.1. Introduction 41
2.2. Logique modale des propositions 43
2.2.1. Introduction 43
2.2.2. Syntaxe de la logique modale 44
2.2.3. Sémantique de la logique modale 45

2.2.4. Interprétations de la logique modale 46
2.2.4.1. Logique du possible et du nécessaire 46
2.2.4.2. Logique temporelle 47
2.2.4.3. Logique dynamique 47
2.2.4.4. Logiques de croyance et de connaissance 48
2.2.4.5. Formules remarquables 48
2.2.5. Relations binaires et schémas de formules 48
2.2.6. Systèmes axiomatiques 50
2.2.6.1. Valuations et tautologies 50
2.2.6.2. Logiques 50
2.2.6.3. Logiques normales 51
2.2.6.4. Systèmes axiomatiques 51
2.2.7. Quelques logiques classiques 52
2.2.8. Système S5 : nécessité et connaissance introspective 53
2.2.9. Les logiques obtenues à partir des schémas D, T, B, 4, 5 56
2.2.10. Langages multimodaux 59
2.2.11. Exemples de langages multimodaux 59
2.2.12. La logique dynamique 60
2.2.13. Logique temporelle et logiques de nécessité 61
2.2.14. Logique déontique 62
2.2.14.1. Le système OS de Von Wright 62
2.2.14.2. Le système NS (Von Wright 1964) : approche dyadique 63
2.2.14.3. Réduction à la logique modale aléthique (Anderson 1958) 64
2.2.14.4. Approche dynamique 65
2.3. La sémantique algébrique 65
2.3.1. Introduction 65
2.3.2. La logique à trois valeurs de Lukasiewicz 66
2.3.3. Les logiques multivaluées 67
2.3.4. Sémantiques pour des logiques à quatre valeurs 68
2.3.5. Logiques multivaluées codées binaires 69
2.3.6. Interprétation ensembliste de logiques multivaluées 71
2.3.7. Structures modales et logiques multivaluées 73

Chapitre 3. Logiques modales temporelle et épistémique 77

3.1. Le temps et le changement 77
3.1.1. Introduction 77
3.1.2. Relations entre temps et changement 78
3.1.3. Le temps : ensemble de points ou d'intervalles 78
3.1.3.1. Les instants comme primitives du temps 79
3.1.3.2. Les intervalles comme primitives du temps 79
3.1.4. Etats et événements 80
3.2. La logique temporelle des propositions 81
3.2.1. Syntaxe 81
3.2.2. Sémantique 82

3.2.3. Axiomatique 83
3.2.4. Logique temporelle linéaire 84
3.2.4.1. Exemples 85
3.2.4.2. Axiomatisation 85
3.2.5. Opérateurs spécialisés 86
3.2.6. Propriétés additionnelles de la relation d'accessibilité 87
3.2.7. Logique temporelle non transitive 89
3.2.8. Logique temporelle à plusieurs dimensions 90
3.2.9. Logique d'intervalles 91
3.3. Logique épistémique 92
3.3.1. Introduction 92
3.3.2. Les mondes possibles en logique épistémique 92
3.3.3. Exemple : Le problème des trois conseillers 94
3.3.4. Connaissance commune et distribuée 96
3.3.5. Exemple : Le problème des trois conseillers (suite) 97
3.3.6. Connaissance et temporalités 101

Chapitre 4. Logique modale quantifiée 103

4.1. Introduction 103
4.2. Syntaxe et sémantique 104
4.2.1. Syntaxe 104
4.2.2. Sémantique 104
4.2.3. Extension et intension 106
4.2.4. Désignateurs rigides et non rigides 107
4.2.5. Les choix en logique modale quantifiée 108
4.3. Interprétation objectuelle 110
4.3.1. Termes rigides; domaines fixes 111
4.3.2. Termes rigides; domaines dépendant du monde 113
4.3.2.1. Le système Q1R 113
4.3.2.2. Remarques 114
4.3.3.Désignateurs non rigides 116
4.3.3.1. Le système Q3 116
4.3.3.2. Le système Q3L 117
4.3.3.3. Application à la logique aléthique 117
4.4. Domaines de quantification contenant des éléments de type intensionnel 118
4.4.1. Introduction 118
4.4.2. Interprétation conceptuelle 118
4.4.2.1. Domaines fixes; le système QC 118
4.4.2.2. Domaines dépendant du monde; le système Q2 119
4.4.3. Interprétation substantielle 119

Chapitre 5. Logiques temporelle et épistémique quantifiées 121

5.1. Logique temporelle quantifiée 121
5.1.1. Introduction 121
5.1.2. Quantification et modalités temporelles 123
5.1.3. Version quantifiée de logiques normales 125
5.1.4. Changement temporel, identité, loi de Leibniz 127
5.1.5. Logique modale quantifiée et temps arborescent 127
5.2. Logique épistémique quantifiée 128

Chapitre 6. Théorie des modèles. Première partie 131

6.1. Introduction 131
6.2. Affinement de la notion de conséquence en logique modale 133
6.3. Validité en logique modale 134
6.3.1. Vérité et validité 134
6.3.2. Validité dans un modèle 134
6.3.3. Validité dans une classe de modèles 134
6.3.4. Validité dans une structure 135
6.3.5. Valuation et substitution 136
6.3.6. Utilité de conserver la notion de validité dans une classe de modèles 138
6.4. Relations de conséquence logique 139
6.4.1. La conséquence logique en logique non modale 139
6.4.2. Les variables au sens conditionnel et au sens de la généralité 139
6.4.3. Force des deux relations de conséquence en logique du premier ordre 141
6.4.4. La conséquence locale 142
6.4.5. La conséquence globale pour les modèles 143
6.4.6. La conséquence globale pour les structures 143
6.4.7. Force des trois relations de conséquence en logique modale 143
6.4.8. Importance de la conséquence locale 144
6.4.9. Conséquence globale sur les structures et règle d'échange 146
6.4.10. Le lien unissant la conséquence locale à la conséquence globale 147
6.5. Le théorème de la déduction en logique modale 148
6.5.1. Le théorème de la déduction en logique modale et en logique non modale 148
6.5.2. Les deux sortes de prémisses 150
6.5.3. Systèmes non normaux 150
6.5.4. Le théorème de la déduction revisité 151
6.5.5. Le théorème local de la déduction 151
6.5.6. Le théorème global de la déduction 152
6.6. La preuve de complétude par la méthode du modèle canonique 152
6.6.1. Correction, complétude, caractérisation pour les systèmes sans prémisses 152

6.6.2. Caractérisation 153
6.6.3. La complétude prouvée à partir du modèle canonique 153
6.6.4. La structure canonique 154
6.6.5. La valuation canonique 155
6.6.6. Le modèle canonique 155
6.6.7. Le système canonique 155
6.6.8. La preuve de complétude 156
6.6.8.1. Introduction 156
6.6.8.2. Définitions préliminaires 156
6.6.8.3. Propriétés de clôture des ensembles maximaux consistants . 156
6.6.8.4. Equivalences remarquables 157
6.6.8.5. Lemme d'existence (Lemme de Lindenbaum) 158
6.6.8.6. Le modèle canonique 158
6.6.8.7. Lemme de dérivabilité 159
6.6.8.8. Lemme préparatoire à l'extension de la valuation des atomes aux formules 160
6.6.8.9. Le lemme d'extension de la valuation V en V' 161
6.6.8.10. Le théorème de complétude 162
6.6.8.11. Adéquation faible et adéquation forte 162
6.6.8.12. Application de la méthode de preuve par les modèles canoniques à d'autres systèmes 163

Chapitre 7. Théorie des modèles. Deuxième partie 165

7.1. Filtration et modèles finis 165
7.1.1. La filtration 165
7.1.2. Notions préliminaires 165
7.1.3. Théorème de préservation de la valuation 167
7.1.4. Préservation de la non-validité 168
7.1.5. La propriété du modèle fini (p.m.f.) 168
7.1.6. La propriété de la structure finie (p.s.f.) 169
7.1.7. Modèles finis, structures finies et décidabilité 169
7.2. La théorie de la correspondance 170
7.2.1. La traduction de la logique modale propositionnelle dans le calcul des prédicats 170
7.2.2. Conséquences de la possibilité de traduire la logique modale propositionnelle en logique du premier ordre 171
7.2.3. Limites de la traduction dans la logique du premier ordre 171
7.2.4. Comment établir un résultat de correspondance ? 172
7.2.5. L'algorithme de Sahlqvist-van Benthem 173
7.2.6. Notions préliminaires 173
7.2.7. La traduction en logique du deuxième ordre 174
7.2.8. L'instantiation minimale 174
7.2.9. Exemple d'application de l'algorithme 175
7.2.10. Justification de l'algorithme 176

7.2.11. Portée et limites de l'algorithme 177
7.3. Résultats de préservation 178
7.3.1. Les sous-modèles engendrés 178
7.3.2. Définitions préliminaires 178
7.3.3. Intérêt de la notion de modèle et de sous-modèle engendrés 179
7.3.4. Théorème de préservation pour les sous-modèles engendrés 179
7.3.5. Ultrafiltres et ultraproduits 180
7.3.6. Théorème de Loš 182
7.3.7. Autres résultats de préservation 182

Chapitre 8. Théorie de la preuve pour la logique modale propositionnelle 185

8.1. Introduction 185
8.2. Méthode de déduction naturelle 186
8.2.1. Les règles et notions nouvelles 186
8.2.2. Exemple de démonstration en déduction naturelle 188
8.3. La méthode des tableaux explicites 191
8.3.1. Définition 191
8.3.2. La preuve par tableaux explicites 191
8.3.3. Les règles d'élimination des opérateurs modaux 191
8.3.4. Règles duales 192
8.3.5. Tableau récapitulatif des règles 194
8.3.5.1. Règle alpha de prolongation 194
8.3.5.2. Règle bêta de ramification 194
8.3.5.3. Règle pi de possibilité 194
8.3.5.4. Règle nu de nécessité 194
8.3.6. Exemple de preuve dans K 195
8.3.7. Les règles qui diffèrent selon les systèmes modaux 195
8.3.8. La part du choix 196
8.3.9. Exemple de démonstration dans S4 196
8.3.10. Tableaux avec prémisses locales et tableaux avec prémisses globales 197
8.4. La méthode des tableaux implicites 198
8.4.1. Le problème de la décision dans le calcul des propositions et dans le calcul des prédicats non modaux 198
8.4.2. Preuve constructive de la complétude 198
8.4.3. La méthode des tableaux implicites 199
8.4.4. La forme des règles dans la méthode des tableaux implicites 199
8.4.5. Les règles structurelles 200
8.4.6. La règle de coupure 200
8.4.7. Définitions préliminaires 200
8.4.8. Relâchement du principe de la sous-formule 202
8.4.9. La propriété de la sur-formule analytique 202
8.4.10. Restriction sur la coupure 203

8.4.11. Règles de réduction 203
8.4.12. Exemples de démonstration 204
8.5. Correction, complétude et décidabilité 205
8.5.1. Caractère non déterministe de la procédure 205
8.5.2. Preuve de la correction de R. Goré 206
8.5.3. La notion d'ensemble saturé vers le bas 207
8.5.4. Vue d'ensemble sur la preuve de complétude de R. Goré 209
8.5.5. La procédure de saturation vers le bas est effective 210
8.5.6. Construction du graphe de modèle 210
8.5.7. Lemme d'existence d'un modèle 210
8.6. La méthode des graphes cycliques orientés avec racine 212
8.7. La méthode des diagrammes 213
8.7.1. Le test de validité dans la logique classique 213
8.7.2. Le test des diagrammes pour K 213
8.7.3. Le test des diagrammes pour T 215
8.7.4. Le test des diagrammes pour S4 216
8.7.5. Méthode de preuve par réfutation et méthode de décision 217
8.7.6. La méthode des tableaux pour la logique temporelle 218
8.7.7. Incomplétude de la logique temporelle linéaire du premier ordre 220

Chapitre 9. Théorie des modèles et théorie de la preuve pour la logique modale quantifiée 221

9.1. Introduction 221
9.1.1. La formule de Barcan et sa converse 223
9.1.2. Sémantique de la logique modale quantifiée avec domaine variable 223
9.1.3. Modèles falsificateurs pour la formule de Barcan 224
9.1.4. Modèle falsificateur pour la converse de la formule de Barcan 225
9.1.5. Les mérites respectifs des modèles avec domaine constant et des modèles avec domaines variables 226
9.1.6. Immersion de la sémantique à domaines variables dans la sémantique à domaine constant 227
9.1.7. Immersion de la sémantique à domaine constant dans la sémantique à domaines variables 227
9.1.8. Evaluation des deux types d'unification des sémantiques 228
9.1.9. Expression syntaxique de la monotonie et de l'antimonotonie 228
9.2. La preuve par tableaux en logique modale quantifiée 229
9.2.1. Règles de réduction pour les opérateurs modaux 229
9.2.2. Règles de réduction pour les quantificateurs 230
9.2.3. Exemple de démonstration en QK 231
9.2.4. Exemple de non-théorème de QK 231
9.3. Correction et complétude 232

9.3.1. Généralités 232
9.3.2. Extraction d'un modèle falsificateur 232
9.4. Logique modale quantifiée avec identité 233
9.4.1. Règles de réduction pour la logique de l'identité 235
9.5. La preuve de complétude de K quantifié avec identité 235
9.6. Logique modale quantifiée avec prédicat d'existence 236
9.7. La logique des termes singuliers en logique modale quantifiée 238
9.7.1. Le manque d'expressivité du formalisme standard 238
9.7.2. L'introduction des abstraits de prédicats 239
9.7.3. La syntaxe des abstraits de prédicats 240
9.7.4. La sémantique des termes et de la prédication à l'aide d'abstraits de prédicats 240
9.7.5. Les désignateurs rigides 241
9.7.6. Identité nécessaire et coréférentialité 242
9.8. L'argument du lance-pierres 242
9.8.1. Transparence et opacité 242
9.8.2. Contextes intensionnels 243
9.8.3. L'anéantissement des contextes intensionnels 244
9.8.4. Une version stylisée de l'argument du lance-pierres 244
9.8.5. Réfutation du lance-pierres 245
9.8.6. La réhabilitation de la logique modale quantifiée 245
9.8.7. Logique modale quantifiée et déduction naturelle 247
9.8.8. Implémentation de la logique modale quantifiée 247

Chapitre 10. Introduction à l'intelligence artificielle 249

10.1. Une définition naturelle de l'intelligence artificielle 250
10.1.1. Qu'est-ce qu'un comportement intelligent ? 250
10.1.2. Simuler un comportement intelligent 252
10.1.3. Intelligence artificielle et informatique 253
10.1.4. Le point de vue anthropomorphe 253
10.1.5. Intelligence et traitement de l'information 255
10.2. Une définition orientée de l'intelligence artificielle 258
10.2.1. Qu'est-ce qu'un programme ? 258
10.2.2. Qu'est-ce qu'un système intelligent ? 261
10.2.3. Les Tours de Hanoï 262
10.2.4. Nim 267
10.2.5. Echecs et Hexapion 268
10.3. Pour en savoir plus 271

Chapitre 11. Principes de la programmation logique 273

11.1. Concepts de base 273
11.1.1. Les termes 273
11.1.2. Faits et règles 276

11.1.3. Questions 278
11.2. Sémantique 280
11.2.1. Sémantique opérationnelle 280
11.2.2. Sémantique déclarative 284
11.3. Approche théorique de la programmation logique 285
11.3.1. La programmation logique propositionnelle 286
11.3.2. PROLOG propositionnel 290
11.4. Programmer en PROLOG 290
11.4.1. Listes, préfixes, suffixes, sous-listes et sous-ensembles 290
11.4.2. Machines abstraites et fonctions récursives 296
11.4.2.1. Automates 296
11.4.2.2. Fonctions récursives 299
11.4.3. Calcul symbolique 302
11.5. Pour en savoir plus 304

12 Le langage PROLOG en intelligence artificielle 305

12.1. Enigmes attribuables à Lewis Carroll 306
12.1.1. Un problème d'intelligence artificielle ? 307
12.1.2. Un programme de résolution en PROLOG 308
12.1.3. La classe de mathématiques 310
12.1.4. Les aléas de l'exploration 314
12.1.5. Une enquête de Sherlock Holmes 317
12.1.6. L'énoncé 318
12.1.6.1. Une solution directe 319
12.1.6.2. Une solution plus rapide 321
12.1.6.3. PROLOG comme outil de preuve 322
12.2. Une limitation importante 322
12.2.1. Un tri "orienté IA" 323
12.2.2. Vers un tri raisonnablement efficace 324
12.2.3. Approche directe du problème de tri 325
12.2.4. Le tri rapide 326
12.2.5. Le problème des partitions 326
12.2.6. Conclusion 329
12.3. Un exemple d'apprentissage 329
12.3.1. Introduction 329
12.3.1.1. Hexapion 329
12.3.1.2. Principe simple d'apprentissage 330
12.3.2. Le programme de jeu 330
12.3.2.1. Quelques mécanismes particuliers 330
12.3.2.2. Le principe du jeu 331
12.3.2.3. Les coups possibles 332
12.3.3. Le dialogue avec l'adversaire 333
12.3.3.1. Gestion de l'apprentissage 334
12.3.3.2. Prédicats auxiliaires supplémentaires 335

12.3.4. Exemples d'exécution 336
12.3.4.1. Première partie 336
12.3.4.2. Deuxième partie 338
12.3.4.3. Progression de l'apprentissage 338
12.3.4.4. Conclusion 340
12.4. Pour en savoir plus 341

chapitre 13. Vérification assistée par ordinateur 343

13.1. Introduction 343
13.2. Construction d'invariant 347
13.2.1. Une méthode théorique 347
13.2.2. Application : programme séquentiel 349
13.2.3. Application : programme parallèle 351
13.3. Construction de programme 354
13.3.1. Un protocole de transfert de données 355
13.3.1.1. La version initiale 355
13.3.1.2. Perte de messages 355
13.3.1.3. Plus de parallélisme 356
13.3.2. Le principe de la fenêtre de Stenning 358
13.3.2.1. L'idée générale 358
13.3.2.2. Le développement 359
13.3.3. Représentation des processus 360
13.3.3.1. Identification des processus 360
13.3.3.2. Communication asynchrone 361
13.3.4. Amélioration du parallélisme 363
13.3.5. Conclusion 363
13.4. Validation des conditions de vérification 363
13.4.1. Introduction 363
13.4.2. Propriétés structurelles des conditions de vérification 365
13.4.3. Le problème de la validation 367
13.4.4. Type, type réduit et polarité 367
13.4.5. Rang d'une sous-formule 368
13.4.6. Polarité et interpolation 369
13.4.7. Pertinence et règle d'élimination 370
13.4.8. La règle de connectivité 370
13.4.9. Un exemple 371

Bibliographie 373

Index 389

Avant-propos

L'ouvrage se veut introductif et ne suppose du lecteur que la connaissance préalable de la logique des propositions et de la logique des prédicats, connaissance disponible dans le premier volume ou dans tout autre traité de logique. Le chapitre 13 suppose aussi la connaissance du deuxième volume. Les références occasionnelles à un passage du premier ou du second volume sont respectivement précédées du chiffre romain I ou du chiffre romain II.

Seul le premier chapitre a vocation de résumé historique et a fait l'objet d'une étude bibliographique. Lorsque deux dates figurent dans une référence, la première est celle de la parution, la seconde celle de la version utilisée (réédition ou traduction).

La connaissance analytique réalise la capacité de l'être humain à comprendre le fonctionnement réel des choses par analyse, déduction et démonstration. La connaissance analytique et le raisonnement qui y est associé sont cristallisés, chez l'homme, autour de deux pôles fondamentaux : le raisonnement numérique ou mathématique (par exemple : $2 + 3 = 5$, et : $(x + y)^2 = x^2 + y^2 + 2xy$) et le raisonnement symbolique (par exemple : Tous les hommes sont mortels. Or Socrate est un homme. Donc Socrate est mortel). Ces deux pôles du raisonnement humain sont respectivement assurés, en informatique, par *l'informatique numérique* et *l'intelligence artificielle*. La première rend traitable par ordinateur le raisonnemnt mathématique tandis que la seconde gère principalement le raisonnement symbolique. C'est au rôle joué par la logique en intelligence artificielle que ce volume est consacré.

Le travail du chercheur en intelligence artificielle est de concevoir les concepts et de créer les outils qui permettront - autant que faire se peut – de simuler le raisonnement humain. Parmi les théories formelles qui servent de fondement

pour concevoir ces concepts et valider ces outils, la logique – et particulièrement la *logique modale* – occupe une place privilégiée.

Le chapitre 2 est consacré à un exposé introductif de la logique modale propositionnelle dans son rôle de représentation de la connaissance et de support du raisonnement. Le chapitre 3 traite des logiques temporelle propositionnelle et épistémique propositionnelle. Les chapitres 4 et 5 sont respectivement consacrés aux logiques modale prédicative, temporelle prédicative et épistémique prédicative. Les chapitres 6 et 7 développent la théorie des modèles tandis que les chapitres 8 et 9 considèrent la théorie de la preuve respectivement pour la logique modale propositionnelle et pour la logique modale quantifiée.

Si la logique a pour vocation de formaliser le raisonnement, l'informatique a celle de l'automatiser. La *programmation logique* est la mise en œuvre informatique du raisonnement; c'est en quelque sorte la logique en action.

Le chapitre 10 est une introduction générale à l'intelligence artificielle, focalisée sur ses liens avec la logique. Les chapitres 11 et 12 développent certains aspects de la programmation logique; en particulier, on y aborde le langage de programmation PROLOG, outil incontournable en intelligence artificielle. Le chapitre 13 est consacré à la vérification automatique des propriétés des programmes parallèles.

Chapitre 1
Introduction historique et philosophique à la logique modale

1.1. L'étude des modalités relève-t-elle de la logique ?

Dans *Mathematical Logic*, Quine distingue les propositions logiquement vraies des autres propositions par le fait que leur valeur de vérité repose uniquement sur des particules de base et qu'elle est indépendante des autres ingrédients qui la composent : "Un énoncé *logiquement vrai* – écrit-il – a cette particularité que les particules telles que 'est', 'ne pas', 'et', 'ou', 'à moins que', 'si', 'alors', 'quelque', 'tout', etc. y figurent de manière telle que cet énoncé soit vrai indépendamment de ses autres ingrédients" [Quine 40, 81], p. 1.

Par exemple, l'énoncé : "Si tout *humain* est *mortel* et *Socrate* est *humain*, alors *Socrate* est *mortel* ", est une vérité de logique. En effet, cet énoncé est vrai et le reste pour toute substitution uniforme effectuée sur les termes qui n'appartiennent pas au vocabulaire logique constitué par la liste des particules. Par exemple, il reste vrai si l'on remplace respectivement "humain" par "planète", "mortel" par "corps céleste" et "Socrate" par "Uranus".

La définition quinéenne de la "vérité de logique" repose sur celle de "vocabulaire logique", qui, à son tour, est défini par énumération. L'énumération donnée ci-dessus n'est pas exhaustive; la question se pose donc de savoir quels termes on peut ajouter à cette liste de particules. A cette question, Quine répond à la deuxième page de l'ouvrage précité : "Les particules qui constituent le vocabulaire logique sont communes à tous les discours. Contrairement aux sciences particulières qui ont un domaine particulier, la logique a un sujet universel (*logic has a universal subject matter*)".

Cette manière de délimiter le vocabulaire logique ne va pas sans problème, mais comme il s'agit d'une définition consacrée par la tradition, nous l'adopte-

rons. Elle permet de répondre affirmativement à la question figurant dans le titre de cette section : *L'étude des modalités relève-t-elle de la logique ?* On peut en effet légitimement ajouter à la liste des particules de base "possible", "nécessaire", "impossible", termes qui appartiennent à tous les discours. L'énoncé : "S'il est possible que *Socrate soit assis* alors il n'est pas impossible que *Socrate soit assis*", reste vrai pour toute substitution uniforme des termes non logiques (les phrases en italique dans le texte).

1.2. Le carré modal

Dans son traité *De l'interprétation*, Aristote expose ce qu'on a appelé par la suite le "carré logique". Ce carré expose les relations logiques (contradiction, contrariété, subcontrariété) existant entre les énoncés instanciant les quatre formes fondamentales "tout A est B ", "quelque A n'est pas B ", "aucun A n'est B " et "quelque A est B ". Il codifie les inférences valides (et les vérités de logique correspondantes) que l'on peut former avec deux énoncés construits à partir des formes fondamentales (voir I, p. 23, 49).

Dans le même traité, Aristote expose les relations logiques existant entre "il est possible que A soit B ", "il est nécessaire que A soit B ", "il est impossible que A soit B ". Au Moyen-Age, on a résumé ces relations par un carré modal calqué sur celui modélisant des relations logiques; ce carré est représenté à la figure 1.1.

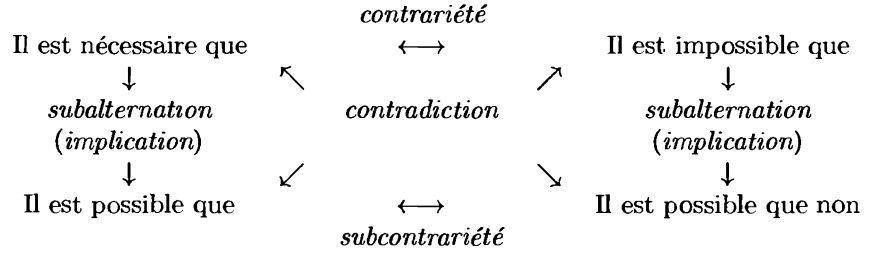


Figure 1.1 *Carré modal*

Cette présentation didactique simplifie quelque peu la réalité historique en ne retenant qu'un des deux sens qu'Aristote donne au mot "possible", à savoir le sens moderne. Or le mot "possible" est aussi entendu par Aristote dans un second sens, celui de "contingent" (c'est-à-dire soumis au hasard, accidentel, éventuel, fortuit). C'est ce deuxième sens qui est à l'œuvre dans le passage suivant : "...de cette proposition (*il est possible que cela soit*), découle une double possibilité (*être ou ne pas être*)" ([Aristote 59], 22b15-20). Nous utiliserons le mot "contingent" pour désigner ce second sens du mot "possible". Si l'on prend "possible" dans le sens de "contingent", le carré modal se réduit à un triangle modal, comme J. Gardies le note dans son *Essai sur la logique des*

modalités : "Les trois modalités retenues sont ici incompatibles deux à deux" [Gardies 79], p. 18. Ce triangle modal est représenté à la figure 1.2.

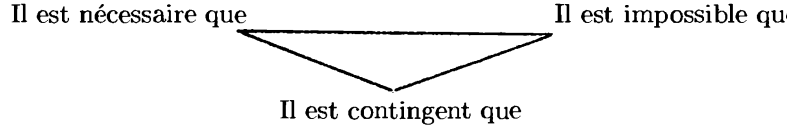


Figure 1.2 *Triangle modal*

1.3. Le syllogisme modal

La philosophie (contemporaine) des sciences distingue la *généralité nomologique*, que l'on trouve dans les énoncés de loi tels que : "Tous les aimants attirent la limaille de fer", des *généralités accidentelles* telles que : "Toutes les personnes présentes dans cette salle sont bilingues".

C'est à une autre distinction que fait appel Aristote pour caractériser les propositions scientifiques. Pour Aristote, la distinction de base se situe entre *propositions nécessaires* et *propositions contingentes* – ou *accidentelles*.

Dans *Les seconds Analytiques*, Aristote pose en principe que la démonstration a pour objet une conclusion nécessaire et exige que le syllogisme qui y conduit ait des prémisses nécessaires. Il fait une distinction entre les prémisses universelles qui sont *simplement vraies* et les prémisses qui sont *nécessaires* en ce sens que les attributs appartiennent nécessairement à leur sujet; il appelle ces propositions "apodictiques". Aristote emploie l'expression "par soi" pour désigner cette appartenance nécessaire : "Sont *par soi*, en premier lieu, les attributs qui appartiennent à l'essence du sujet; c'est ainsi qu'au triangle appartient la ligne, et à la ligne le point. En second lieu, ce sont les attributs contenus dans des sujets qui sont eux-mêmes compris dans la définition exprimant la nature de ces attributs" [Aristote 62], 73a30-35.

Pour élaborer une *science démonstrative*, il faudra donc remplir trois conditions :

1. Partir de prémisses nécessaires.
2. Arriver à une conclusion nécessaire.
3. Passer des prémisses aux conclusions par des syllogismes valides.

Comme les syllogismes de la science démonstrative doivent être constitués de deux prémisses *nécessaires*, ils sont un sous-ensemble de la classe des *syllogismes modaux*, classe qui contient aussi les syllogismes dont l'une ou les deux prémisses sont contingentes.

Dans *Les premiers Analytiques*, sections 8 à 22, Aristote élabore une théorie du syllogisme modal [Aristote 47]. La théorie aristotélicienne des syllogismes dont au moins une prémisse est *contingente* contient des erreurs. Ce n'est pas

le lieu ici de les recenser. Il importe, en revanche, de relever une distinction importante entre deux sortes de nécessité qu'on trouve chez Aristote, bien qu'il ne l'ait pas thématisée; il s'agit de la distinction entre *nécessité logique* et *nécessité ontique*.

Soit le syllogisme suivant mentionné par W. et M. Kneale dans *The Development of Logic* :

> Tout animal est *nécessairement* mortel.
>
> Tout homme est *nécessairement* un animal.
>
> Donc nécessairement tout homme est *nécessairement* mortel.

"Selon la théorie aristotélicienne l'adverbe *nécessairement* à l'intérieur des prémisses (en caractères italiques dans notre texte) marque la connexion essentielle unissant l'animalité, la mortalité et l'humanité" [Kneale and Kneale 62], p. 95. Deux sortes de nécessités sont à l'œuvre dans ce syllogisme :

1. La nécessité de la connexion entre propriétés. Nous l'appellerons *nécessité ontique* puisqu'elle concerne l'*être* (*on* en grec).
2. La nécessité de la connexion logique liant la vérité de la conclusion à celle des prémisses dans tout syllogisme valide. Nous l'appellerons *nécessité logique* (en caractères télétypes dans notre texte).

Comme nous le constatons, il y a indiscutablement plusieurs sortes de nécessités et, corrélativement, d'impossibilités. L'impossibilité de dénombrer les nombres réels est d'une autre nature que l'impossibilité de dépasser la vitesse de la lumière. Nous renvoyons le lecteur intéressé par ces questions à *Quand la science a dit... c'est impossible* [Alimi et al. 99]. On verra au chapitre 2 comment la logique modale contemporaine permet de formaliser ces différentes sortes de nécessité.

1.4. Les futurs contingents

Dans le traité *De l'Interprétation*, Aristote développe un argument qui vise à prouver que si la loi de la bivalence (toute proposition est vraie ou fausse) s'applique aux propositions futures portant sur des singuliers telles que, par exemple : "Il y aura une bataille navale demain", alors : "Il n'y a plus place pour le hasard et il n'y a plus ni à délibérer, ni à se donner de la peine. Car délibérer et se donner de la peine n'ont de sens que pour quelqu'un qui croit que s'il accomplit une certaine action, un résultat suivra et que s'il ne l'accomplit pas, le résultat ne suivra pas".

La passage clé d'Aristote que nous allons citer n'est pas facile à comprendre; nous le ferons suivre d'une reconstruction due à S. Haack. Aristote écrit : "Mais s'il était toujours vrai de dire qu'une chose est ou sera, il n'est pas possible qu'elle ne soit pas ou qu'elle ne sera pas; or ce qui ne peut pas ne pas se produire est dans l'impossibilité de ne pas arriver, et ce qui est dans

l'impossibilité de ne pas arriver arrive nécessairement. Il en résulte ainsi que tous les futurs se produisent nécessairement. Par suite, rien n'arrive d'une manière indéterminée ou par l'effet du hasard, car là où il y a hasard, il n'y a pas nécessité" [Aristote 59], 9b10-15.

Łukasiewicz a été convaincu par le raisonnement d'Aristote. Considérant que les énoncés sur les futurs contingents ne sont pas dépourvus de valeur de vérité, à l'opposé des questions, des ordres et des souhaits, il a forgé pour eux une nouvelle valeur de vérité, distincte du vrai et du faux : "Cette valeur, que nous désignons par 1/2, représente le *possible* et s'ajoute comme troisième valeur au *vrai* et au *faux* " ([Łukasiewicz 30], p. 53, cité par [Haack 74], p. 73). R. Ntambue a montré dans *Eléments de logique trivalente* que la logique trivalente peut être interprétée dans une logique modale [Ntambue 98]. Cette interprétation sera aussi présentée à la section 2.3 qui rassemble certaines conclusions relatives aux connexions entre logiques modales et logiques multivaluées (voir aussi [Rescher 69]).

S. Haack juge le raisonnement d'Aristote valide, mais reproche à l'une de ses prémisses d'être non fondée. Elle en déduit que la conclusion du raisonnement n'a pas été établie et que, dès lors, la bivalence de la logique peut être conservée. Elle reconstruit le raisonnement d'Aristote comme suit [Haack 74], p. 73 :

1. Si chaque phrase (dont le verbe) est au futur est soit vraie, soit fausse, alors pour chaque paire consistant en une phrase au futur et en la négation de celle-ci, l'une des phrases est vraie et l'autre est fausse.
2. Si, pour chaque paire consistant en une phrase au futur et en sa négation, l'une des phrases doit être vraie et l'autre fausse, alors tout ce qui arrive, arrive *par nécessité.*
3. Mais tout n'arrive pas *par nécessité.* Certains événements sont contingents.
4. Donc, aucune phrase au futur n'est vraie ou fausse.

C'est la deuxième prémisse qui, selon S. Haack, serait non fondée ou, plus exactement, mal fondée.

1.5. L'école de Mégare

Cette école philosophique a été fondée au cinquième siècle avant J.-C. et se perpétua jusqu'au début du troisième siècle avant J.-C. L'un de ses premiers représentants est Eubulide à qui l'on doit plusieurs paradoxes célèbres tels que le paradoxe du *Menteur* et le paradoxe du *Tas.*

Diodore Cronos enseigna à Mégare dans les dernières années du quatrième siècle avant notre ère. Il est connu pour sa conception de l'implication que critiqua son élève Philon qui défendait en gros ce qu'on appelle aujourd'hui l'implication matérielle. Nous y reviendrons. Il est également connu pour la position qu'il prend à l'égard de l'argument du *Dominateur*, position qui le

conduit à défendre le fatalisme logique et à définir le possible au moyen de modalités temporelles.

L'argument du *Dominateur*, rapporté par Epictète dans la traduction que propose [Barreau 84], p. 757, consiste dans les trois propositions suivantes qui sont incompatibles :

1. Toute vérité qui appartient au passé est nécessaire.
2. Du possible ne suit pas l'impossible.
3. Il y a du possible tel qu'il n'est pas vrai ni ne sera vrai.

Confronté à ce trilemme, le stoïcien Cléanthe accepte les deux dernières propositions et rejette la première. Le stoïcien Chrysippe accepte la première et la troisième. Quant au mégarique Diodore Cronos, il accepte les deux premières et rejette la troisième, souscrivant ainsi au *fatalisme logique* (voir le livre *Nécessité ou contingence*, [Vuillemin 84]).

On notera que Diodore Cronos définit les modalités ontiques au moyen des modalités temporelles; il adopte la définition suivante du possible : *Il est possible que p si et seulement s'il est maintenant vrai ou il sera vrai un jour que p* [Gardies 75], p. 27. Par exemple : "Il est possible que je sois à Corinthe" signifie : "Je suis à Corinthe ou j'y serai un jour". Corrélativement, la définition du *nécessaire* sera : *Il est nécessaire que p si et seulement s'il est maintenant vrai et sera toujours vrai que p.* Gardies observe que si l'on adopte la définition diodoréenne du possible conjointement à la conception linéaire du temps (le passé est unique et le futur est unique), on aboutit à un déterminisme intégral. Les modalités temporelles telles qu'elles ont été définies par les écoles stoïque et mégarique seront étudiées aux paragraphes 2.2.13 et 5.1.2.

1.6. Les modalités temporelles au Moyen Age

Cherchant à échapper au fatalisme logique auquel aboutit Diodore Cronos, le philosophe franciscain anglais Guillaume d'Ockham (1285-1347) nie, comme le faisait déjà Cléanthe, que les propositions au passé soient toutes nécessaires. Mais, comme le note Prior dans *Past, Present and Future*, Ockham ajoute, aux propositions au passé qui ne sont pas équivalentes à des propositions au futur, le principe selon lequel *ce qui a été ne peut pas maintenant ne pas avoir été.* Pour reprendre l'exemple de Prior, la proposition : "Il était le cas hier qu'il serait le cas dans deux jours que je-suis-en-train-de-fumer" ne jouit pas de l'irrévocabilité du passé. En effet elle est équivalente à la proposition au futur : "Il sera le cas demain que je suis en train de fumer" [Prior 67], p. 121.

La restriction faite par Ockham est bienvenue, mais elle n'est pas assez précise. On veut exclure la proposition : "Il était le cas *hier* qu'il serait le cas *deux jours plus tard* que je suis en train de fumer" des propositions privilégiées (dont la vérité est irrévocable); mais on souhaite accueillir parmi ces dernières la proposition : "Il était le cas *hier* qu'il serait le cas *demain*

que je suis en train de fumer". La difficulté, c'est d'exclure *d'une manière systématique* les propositions qui, en raison d'un futur emboîté dans un passé, sont des propositions qui parlent de l'avenir, et d'inclure les autres. Prior a réussi à surmonter cette difficulté et à formaliser les intuitions d'Ockham dans un système axiomatique qu'il appelle *Ot* (logique temporelle ockhamiste). Cette logique contient parmi ses axiomes le principe : "Nécessairement p implique toujours dans le futur p", principe que Prior a trouvé dans le traité *de Predestinatione* d'Ockham [Copeland 96], p. 21.

Sans entrer dans les détails, disons seulement que la *sémantique* de ce système ockhamiste exige le recours à une structure dans laquelle le temps n'est plus représenté par une flèche (ou temps linéaire), mais bien par une fourche à deux dents. Le temps est conçu comme *ramifié* vers l'avenir. Dans l'*Essai sur les données immédiates de la conscience* de 1888, Bergson envisage un temps ramifié vers l'avenir pour rendre compte de la liberté, mais rejette aussitôt cette représentation *spatiale* du temps comme trompeuse. La première description détaillée d'une représentation ramifiée du temps, au vingtième siècle, apparaît dans une nouvelle de J. Borges parue en 1941 ([Øhstrøhm and Hasle 95], p. 181). Notons enfin que nous étudions la notion de temps ramifié dans les paragraphes 3.2.6 et 5.1.5.

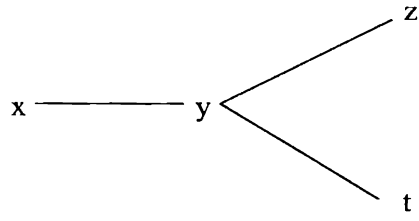


Figure 1.3 *Représentation spatiale du temps ramifié*

En donnant des valeurs de vérité convenables aux atomes propositionnels, il est possible de faire de cette *structure* (la fourche) un *modèle* qui vérifie au point y, pour une des routes d'évolution du monde, la proposition : "Il y aura une bataille navale demain", mais qui ne vérifie pas la proposition : "Nécessairement il a été vrai qu'il y aura une bataille navale demain" (voir la figure 1.3). Le temps ramifié (encore qualifié d'*arborescent*) vers l'avenir permet donc, sans sacrifier la *loi de bivalence*, de maintenir la croyance que l'avenir est ouvert, contrairement au passé et au présent.

On se souvient que parmi les prémisses de l'argument par lequel Aristote prouvait l'indétermination du futur figurait le principe : "Toute chose qui est, est nécessairement, quand elle est". La traduction dans une formule de la logique modale ontique donne : "p implique *nécessairement* p", ce qu'on représente formellement par la formule $p \supset \Box p$ (où l'opérateur modal $\Box$ modélise *nécessairement*). Cette formulation en *logique modale ontique* du

principe aristotélicien est inadéquate. D'abord parce qu'elle ne capte pas ce qu'a voulu dire Aristote. En effet, si dans cette formule on substitue Fp (il sera le cas dans le futur que p) à p, on obtient $Fp \supset \Box Fp$, qui formalise l'affirmation du *fatalisme* : "Tout ce qui arrivera, arrivera nécessairement". Ensuite, parce que si on combine la formule $p \supset \Box p$ avec l'axiome qui dit que "ce qui est nécessairement le cas est le cas", c'est-à-dire avec $\Box p \supset p$, on obtient l'équivalence suivante qui anéantit la modalité : $\Box p \equiv p$. La solution que préconise P. Bailhache dans *Essai de logique déontique* consiste à *temporaliser* la proposition au moyen de l'opérateur temporel $R_t p$ qui se lit : "Il est réalisé au temps t que p". On représentera alors le principe invoqué par Aristote par $R_{t'}p \supset R_t p \Box R_{t'}p$ où t' est une date antérieure à t. On peut montrer que cette dernière proposition "est vraie lorsque $R_{t'}p$ porte vraiment sur le passé de t" [Bailhache 91], p. 76. Notons que le lexique de ce système ne contient pas les opérateurs modaux classiques P (il a été le cas dans le passé) et F (il sera le cas dans le futur).

Si l'on s'assigne pour tâche d'expliquer pourquoi le raisonnement :

Georges V a épousé la reine Mary;

la reine Mary est veuve;

donc Georges V a épousé une veuve,

est non valide alors que le raisonnement :

A a épousé B,

B est une métisse,

donc A a épousé une métisse,

est valide, on n'a nullement besoin d'une logique des temps grammaticaux et l'on peut se rallier à l'opinion de Quine : "Une logique du temps grammatical est une énorme trivialité que nous n'avons aucune excuse à ériger si notre souci est l'usage du langage scientifique plutôt que l'étude scientifique du langage" [Quine 72], p. 453. Il est vrai que pour décrire la nature, nous pouvons traiter le temps comme une quatrième dimension, mais si nous voulons rendre compte de l'homme dans sa capacité d'*agent*, par opposition à l'homme comme *objet*, nous avons besoin d'une logique de l'*action* qui inclut une logique du *temps*.

1.7. La logique déontique au Moyen Age

Si la logique modale ontique et la logique modale temporelle apparaissent dès l'Antiquité, la logique déontique, en revanche, ne commence pas avant le treizième siècle. Cette logique concerne les modalités de l'*obligatoire*, du *permis* et de l'*interdit*. La raison de cette apparition tardive a été bien expliquée par S. Knuuttila. Dans l'Antiquité, certains philosophes ont fait place au hasard et à l'indétermination, mais aucun ne fait droit à une volonté libre qui, dans des actes individuels, choisirait non seulement ses moyens, mais aussi ses fins.

Chez Aristote par exemple, l'action humaine est le fruit d'un calcul de la raison utilisant le syllogisme pratique [Gochet 97]. Il n'y a pas de choix des fins. L'homme bon est celui qui réalise parfaitement l'essence de l'homme. L'homme poursuit les fins qui découlent de son essence, avec le secours indispensable de l'éducation.

Ce déterminisme rationnel sera contesté au treizième siècle par les penseurs franciscains Robert de Bruges, John Peckham et Peter Olivi notamment : "S'il appartient à l'essence de l'homme de pouvoir librement choisir le genre d'homme qu'il veut être, il n'est plus possible de dériver de son essence ce qu'il doit être" [Knuuttila 81], p. 235-236. Un nouveau paradigme moral apparaît. L'identité personnelle ne se définit plus en termes de la réalisation d'une essence, mais par le respect d'une règle que l'on peut transgresser. Les termes axiologiques "bon", "indifférent", "mauvais" perdent leur primauté au profit de termes déontiques tels que "obligatoire", "permis", "interdit".

Au quatorzième siècle, Guillaume d'Ockham, Robert Holcot et Roger Rosetus connaissent parfaitement les rapports unissant l'obligatoire ($\bigcirc$), le permis ($[Pe]$) et l'interdit ($[Fo]$), rapports résumés dans les six équivalences logiques suivantes :

(1) $\bigcirc p \equiv \neg[Pe]\neg p$

(2) $[Pe]p \equiv \neg \bigcirc \neg p$

(3) $\neg \bigcirc p \equiv [Pe]\neg p$

(4) $\neg[Pe]p \equiv \bigcirc \neg p$

(5) $\bigcirc p \equiv [Fo]\neg p$

(6) $[Fo]p \equiv \bigcirc \neg p$

La formulation correcte des équivalences qui permettent de traduire les propositions normatives les unes dans les autres est moins simple qu'on ne le pense. Husserl, sur ce point, a commis plusieurs erreurs relevées par Kalinowski. Il affirme, par exemple, que nous pouvons dire indifféremment "A est obligé d'être B", ou, suivant les cas, "A ne peut pas être B". Un peu plus loin, il écrit : "*peut* est la négation de *ne doit pas* ou – ce qui revient au même – de *ne peut pas*" ([Husserl 59], p. 43-44; 1910 pour la première édition).

Les six équivalences de logique déontique qu'on vient de mentionner sont valables aussi en logique modale ontique. Les termes déontiques "obligatoire", "interdit", "permis", ont, entre eux, les mêmes relations que les modalités ontiques "nécessaire", "impossible", "possible". Il ne faut pas en conclure que ces deux logiques sont des variantes terminologiques l'une de l'autre. Nous allons voir que ce n'est pas le cas. Les logiques déontiques diffèrent des logiques modales ontiques par certaines lois et aussi par les structures qui permettent de les modéliser.

En logique modale ontique, on a la règle ($\Diamond$ signifie "possiblement") :

$$\frac{p \supset q}{\Diamond p \supset \Diamond q}.$$

Aristote la mentionnait déjà et la justifiait dans *Les premiers Analytiques* : "...si de l'existence de p suit nécessairement l'existence de q, de la possibilité de p suivra nécessairement la possibilité de q" [Aristote 47], 34a5. Dans son commentaire d'Aristote, Buridan (1298-1358) note que la même règle vaut pour la nécessité ($\Box$ signifie "nécessairement") :

$$\frac{p \supset q}{\Box p \supset \Box q}.$$

La première règle, transposée à la logique déontique, se formule ainsi :

$$\frac{p \supset q}{[Pe]p \supset [Pe]q}.$$

Roger Rosetus, un auteur du quatorzième siècle, montre, au moyen du contre-exemple suivant, que la règle précédente n'est pas valable :

Si quelqu'un se repent de ses fautes, alors il a commis des fautes.

S'il est permis à quelqu'un de vouloir se repentir de ses fautes, alors, il lui est permis de vouloir avoir commis des fautes.

Le conditionnel au-dessus de la barre d'inférence est vrai, mais celui qui est en dessous de la barre d'inférence est manifestement faux.

Le *paradoxe du bon Samaritain*, bien connu en logique déontique contemporaine, est un autre contre-exemple à la règle critiquée par Rosetus. On part de la prémisse :

> Nécessairement si le bon Samaritain aide quelqu'un qui a été volé, alors quelqu'un a été volé.

On invoque ensuite le principe qui a la forme logique suivante :

$$\Box(p \supset q) \supset ([Pe]p \supset [Pe]q).$$

On applique une contraposition sur le conséquent, ce qui donne :

$$\Box(p \supset q) \supset (\neg[Pe]q \supset \neg[Pe]p).$$

Sachant que "non permis" est équivalent à "interdit", on obtient, par *modus ponens*, la conclusion indésirable :

> S'il est interdit que quelqu'un soit volé, alors il est interdit que le bon Samaritain aide quelqu'un qui a été volé.

L'élimination de tels paradoxes est l'une des tâches de la logique déontique. Une interprétation quelque peu différente du paradoxe de bon Samaritain sera proposée au paragraphe 2.2.14 à l'occasion de la présentation de certaines logiques déontiques.

1.8. Les logiques multimodales

Nous venons de voir que les *modalités déontiques* n'obéissent pas aux mêmes lois que les *modalités ontiques*. Il faudra donc constuire des axiomatiques différentes pour capter les différentes sortes de modalités. Il faut, par surcroît, énoncer des axiomes qui expriment les liens unissant les modalités déontiques aux modalités ontiques. Qu'il existe de tels liens est indiscutable. L'adage : "A l'impossible nul n'est tenu" donne, par contraposition : "Devoir implique pouvoir". Ce principe, qu'on exprime formellement par $\bigcirc p \supset \Diamond p$, relie une modalité déontique à une modalité ontique.

Pour exprimer la thèse ockhamiste selon laquelle le passé est inaltérable, mais l'avenir est ouvert, Prior a besoin d'axiomes reliant cette fois les modalités ontiques, telles que "nécessaire", aux modalités temporelles. Dans le système ockhamiste de logique temporelle de Prior (*Ot*), on a les axiomes suivants :

(1) $\Box p \supset Gp$ (G signifie "il sera toujours vrai dans le futur que").

(2) $p \supset \Box Pp$ où aucune formule ne contenant d'occurrence de F (F signifie "il sera au moins une fois vrai dans le futur que", tandis que P signifie "il aura au moins une fois été vrai dans le passé que") n'est substituée à p.

C'est donc à Prior que l'on doit la première logique multimodale, une logique qui a, aujourd'hui, d'importantes applications en théorie de la programmation [Fine and Schurz 96]. Enfin, il y a des systèmes axiomatiques qui comportent les trois sortes de modalités (ontique, déontique et temporelle). Le lecteur est renvoyé à l'ouvrage précité de [Bailhache 91] dans lequel les ressources d'une logique *trimodale* sont utilisées pour formaliser l'action et pour écarter les paradoxes déontiques.

1.9. De l'implication

Dans *Méthodes de Logique*, Quine insiste sur la différence entre le conditionnel et l'implication [Quine 73], p. 53. Dans la phrase : "Si la neige est blanche, alors l'herbe est verte", la particule "si ... alors" est un conditionnel qui connecte deux énoncés. C'est un connecteur du langage-objet. Dans la phrase : "La neige est blanche *implique* l'herbe est verte", le verbe "implique" ne relie pas deux *énoncés*, mais deux *noms d'énoncés*. Le mot "implique" est un prédicat binaire du métalangage.

Le connecteur du conditionnel – et corrélativement l'implication correspondante – peut avoir plusieurs sens. Le sens du conditionnel représenté par le symbole $\supset$ du calcul standard des propositions est donné par une table de vérité (I, p. 97) qu'on peut résumer ainsi : Le conditionnel représenté par $p \supset q$ est faux si l'antécédent est vrai et le conséquent faux. Il est vrai dans les trois autres cas. On peut aussi le définir au moyen de la négation et de la conjonction prises comme termes primitifs :

$$p \supset q =_{def} \neg(p \wedge \neg q).$$

L'implication qui correspond au conditionnel que l'on vient de définir es appelée *implication matérielle.*

Philon de Mégare utilise le conditionnel "si ... alors" dans le sens d l'implication matérielle. Diodore Cronos définit un conditionnel plus fort. Pou lui "si p alors q" est vrai s'il n'est *jamais* le cas que p soit vrai et que q soit faux Enfin, pour Chrysippe, "si p alors q" signifie : "Il est inconsistant que p soi vrai et que q soit faux" [Prior 62], p. 194. On voit poindre ici une *notion modal* dans la définition du conditionnel. L'expression "il est inconsistant" peut, e effet, être comprise comme synonyme de "il est *logiquement impossible* que".

Les ouvrages fondateurs de la logique contemporaine, à savoir l'idéographi de Frege (*Begriffschrift*) remontant à 1879 et les *Principia mathematica* d Whitehead et Russell dont la première édition remonte à 1910-1913, bannissen les modalités de la logique. Frege affirme ce qui suit dans sa *Begriffschrift* "En disant qu'une proposition est nécessaire, je donne une indication sur le fondements de mon jugement. Mais, puisque cela n'affecte pas le conten conceptuel de mon jugement, la forme du jugement apodictique (Frege désign par là des jugements comme *nécessairement* p) n'a aucune signification pou nous" (Frege dans [van Heijenoort 67], p. 13).

1.10. L'implication stricte

En 1918, C. Lewis introduit un conditionnel qui est cette fois-ci indiscutable ment modal [Lewis 18]. On peut le définir ainsi :

$$p \prec q =_{def} \neg\Diamond(p \wedge \neg q),$$

où le définissant de "si p alors q" peut se lire "il est *impossible* que p et non q" L'objectif de Lewis était de définir un conditionnel *plus fort* que le conditionne vérifonctionnel, une implication plus forte que l'implication matérielle. Il y a réussi; chaque fois qu'une formule A implique *strictement* une formule B la formule A implique aussi *matériellement* la formule B, mais la réciproqu n'est pas vraie. Une formule A peut impliquer matériellement une formul B sans l'impliquer strictement. Voici un exemple illustrant la fausseté de l réciproque :

On a (1) : $(p \supset q) \supset \neg(p \wedge \neg q)$

On a (2) : $\neg(p \wedge \neg q) \supset (p \supset q)$

On a (3) : $(p \prec q) \prec \neg(p \wedge \neg q)$

On n'a pas (4) : $\neg(p \wedge \neg q) \prec (p \prec q)$

On peut en effet déduire les formules (1)-(3) du système axiomatique et prouve que la formule (4) n'est pas dérivable des axiomes.

Dans *Symbolic Logic* [Lewis and Langford 32], l'objectif visé par Lewi apparaît en toute clarté. La relation entre p et q que Lewis cherche à cerne au moyen d'un système d'axiomes fixant le sens du connecteur correspondant

c'est *la relation de déductibilité*. En d'autres termes, pour Lewis, "p implique strictement q" doit être pris dans le sens de "q est déductible de p".

Or l'implication matérielle ne peut exprimer la déductibilité. En effet, elle comporte de nombreux théorèmes qui excluent qu'on interprète "p implique matériellement q" comme signifiant "q est déductible de p". Bornons-nous à un seul cas. Le calcul propositionnel classique comporte le théorème suivant : $\neg p \supset (p \supset q)$. Si $p \supset q$ peut se lire "q est déductible de p", le théorème tout entier peut se lire : "Si Non p est vrai, alors de p, on peut déduire n'importe quoi"; ou encore : "Si p est faux, alors de p on peut *déduire* n'importe quoi". Et plus simplement encore : "Du faux on peut *déduire* n'importe quoi". C'est le principe bien connu du *ex falso sequitur quodlibet*. Il est sanctionné par la tradition. Il n'en demeure pas moins qu'il est contre-intuitif. Aucun mathématicien ne s'estimerait autorisé à déduire de la proposition fausse : "La terre est plate", une réponse à une question ouverte comme la question de savoir si la conjecture de Goldbach est vraie.

Lewis n'a pas ce problème car il représente la déductibilité par $-\!\!<$ et non par $\supset$. Or l'énoncé $\neg p -\!\!< (p -\!\!< q)$ n'est pas un théorème de ses systèmes.

1.11. Polysémie de l'implication stricte

Quand on cherche à donner un sens au conditionnel de l'implication matérielle, on dispose de deux méthodes. Comme ce conditionnel est une fonction de vérité, on peut le définir par une table de vérité. D'autre part, on peut le définir à partir de termes primitifs tels que $\wedge$ et $\neg$. Dans le cas de l'implication stricte qui n'est pas une fonction de vérité, la première méthode n'est plus applicable. Il nous reste seulement la seconde, à savoir définir $-\!\!<$ à partir des *termes primitifs* $\wedge$, $\neg$ et $\Diamond$. Mais comment connaître le sens des termes primitifs et spécialement le sens de $\Diamond$? La réponse est que ce sens est défini contextuellement par les axiomes.

Cette réponse n'est pas en soi irrecevable. Cependant, dans le cas de la logique, on est confronté à un problème imprévu; il existe des systèmes d'axiomes non équipollents pour la logique modale, c'est-à-dire des systèmes d'axiomes qui permettent de démontrer des ensembles différents de théorèmes; par exemple, C. Lewis définit cinq systèmes qu'il nomme $S1$ à $S5$. Il existe donc plusieurs sens pour $\Diamond$ et plusieurs sens pour $-\!\!<$. Dès lors la question se pose de savoir s'il existe un système privilégié parmi les systèmes de logique modale. Parmi les conditionnels stricts ($-\!\!<$) de force différente qui figurent dans les systèmes $S1$ à $S5$ de C. Lewis, quel est celui qui représente dans le langage-objet la relation de déductibilité ?

Lewis et Langford reconnaissent le problème et répondent : "Le bon usage qui prévaut dans l'inférence logique – par exemple, la pratique des déductions mathématiques – n'est pas suffisamment précis ni conscient de soi pour déterminer clairement quel est celui de ces cinq systèmes qui exprime les

principes acceptables de déduction" ([Lewis and Langford 32, 59], p. 502, cité par [Rivenc et de Rouilhan 97], p. 14).

On peut, en tout cas, comme le note G. Granger, affirmer que l'implication stricte exprimée dans le langage-objet par le conditionnel $-\!\prec$ est *strictement plus forte* que la relation métalinguistique de déductibilité représentée par le symbole $\vdash$ qui signifie "de ... on peut dériver ...". En effet, comme le dit encore Granger : "Un 'théorème de la déduction' tel que :

$$\frac{a \vdash b}{a -\!\prec b},$$

n'est pas valable dans les systèmes de Lewis" [Granger 95], p. 62.

S'il n'a pas atteint l'objectif visé, Lewis a néanmoins fait une contribution majeure à la logique modale. Les systèmes $S4$ et $S5$ ont été largement utilisés notamment pour formaliser les raisonnements épistémiques dans lesquels "l'agent A sait que" ou "l'agent A croit que" acquièrent le statut d'opérateurs modaux. Par exemple, l'axiome caractéristique de $S4$, à savoir $\Box p \supset \Box\Box p$, où $\Box$ est lu comme "A sait que" permet d'exprimer l'introspection positive : "Si A sait que p, alors A sait qu'il sait que p".

On évoquera au chapitre 3 une énigme (l'énigme des trois conseillers) qui ne peut être résolue que si on raisonne non seulement sur les *données de fait*, mais aussi sur la *connaissance* que certains agents ont de ces données, en utilisant des schémas déductifs où "A sait que" figure comme constante logique.

1.12. La correction des systèmes de Lewis

En 1959, E. Lemmon, que rend perplexe la pluralité des systèmes de logique modale de Lewis, a posé cette question fondamentale : "Existe-t-il un système unique de logique modale qui soit correct ?" [Lemmon 59]. Pour chacun des systèmes qu'il examine, il réussit à trouver une interprétation, chaque fois différente, pour laquelle tous les axiomes du système sont vrais et pour laquelle les règles d'inférence transportent la vérité. Lemmon a donc démontré que des systèmes modaux différents pouvaient être tous corrects.

Prenons, par exemple, le système axiomatique dénommé $SO.5$ (encore dénommé KT; voir § 2.2.7) dont les axiomes sont l'axiome K qui distribue l'opérateur "nécessairement" sur l'antécédent et le conséquent d'un conditionnel (formellement cet axiome se représente par : $\Box(p \supset q) \supset (\Box p \supset \Box q)$), et l'axiome T (si nécessairement p alors p, soit : $\Box p \supset p$), et qui comporte comme règles le *modus ponens*, la règle de *substitution uniforme* et la règle "de p est *une tautologie*, dérivez *nécessairement* p". Lemmon a montré que ce système est correct si on interprète "nécessairement p" comme "p est tautologique".

En revanche, si nous remplaçons la dernière règle d'inférence que l'on vient de citer par la *règle de nécessité* (de p *est un théorème*, dérivez *nécessairement* p *est un théorème*), le théorème "Il est nécessaire que si nécessairement p alors p" cesse d'être vérifié. Car "... quoique ce qui est tautologique soit vrai, il n'est

pas *tautologique* que ce qui est tautologique soit vrai" [Lemmon 59], p. 32. En effet, "tautologique" signifie "vrai en vertu des connecteurs vérifonctionnels". Et ce qui est vrai en vertu des connecteurs vérifonctionnels est vrai tout court. Mais cette dernière affirmation repose elle-même sur le principe : "Ce qui est *vrai en vertu de X* est *vrai* ", principe qui n'a rien à voir avec les fonctions de vérité.

Notons enfin que la correction est une notion relative. Un système axiomatique peut être correct pour une interprétation, sans pour autant l'être pour une autre. Quant aux systèmes de Lewis, ils seront présentés de façon plus détaillée dans les paragraphes 2.2.7, 2.2.8 et 2.3.7.

1.13. L'avènement de la logique modale quantifiée

En 1946 paraissent dans *The Journal of Symbolic Logic* deux articles qui fondent la logique modale quantifiée. Ce sont : "A functional calculus of first order based on strict implication" [Barcan 46], et : "Modalities and quantification" [Carnap 46]. Le premier système repose sur le système $S2$ de Lewis. Le deuxième repose sur un équivalent du système $S5$.

On traitera ici la logique modale quantifiée de Carnap parce que le fragment propositionnel de celle-ci contient une preuve de correction et une preuve de complétude, toutes deux attribuées par Carnap à Wajsberg (1933). C'est la preuve de complétude qu'on examinera.

L'objectif de Carnap est de construire un calcul modal (un système axiomatique non interprété) qui soit correct et complet relativement à l'interprétation de *Nec A* par : "*A* est une vérité de logique". Une proposition est une vérité de logique, – ou, comme le dit Carnap, une proposition est L-vraie dans un système sémantique $\mathcal{S}_i$ – si elle est vraie uniquement en vertu des règles sémantiques de $\mathcal{S}_i$ et indépendamment des faits extra-linguistiques. Observons qu'il importe de distinguer les *systèmes sémantiques* de Carnap, notés $\mathcal{S}_i$, des *systèmes axiomatiques* $S1$–$S5$ de Lewis, que Carnap appelle "calculs".

Prouver la complétude d'une théorie, c'est démontrer que le calcul correspondant est assez puissant pour démontrer tous les énoncés vrais de la théorie. Dans le cas qui nous occupe, la *théorie*, c'est la logique modale propositionnelle, et le *calcul*, c'est le système axiomatique de $S5$ (voir le paragraphe 2.2.8). Démontrer la complétude, c'est prouver que si A est L-vraie dans la logique modale des propositions, alors *Nec A* est démontrable dans le calcul modal des propositions [Carnap 46], p. 46. Carnap montre que c'est bien le cas. Il utilise à cette fin une méthode de réduction à la forme normale, bien connue dans le cas du calcul non modal des propositions, mais qui est étendue aux propositions modales.

Carnap a prouvé également la *correction* d'une *variante quantifiée* de $S5$ pour l'interprétation de "nécessairement A" par "A est L-vraie", mais il n'a pas prouvé la *complétude* de ce calcul modal du premier ordre pour l'interprétation

de la nécessité qu'il utilise. D'autres logiciens, à la fin des années cinquante, fourniront une preuve de complétude de $S5$ quantifié. On ne peut cependant pas dire qu'ils ont *achevé* le travail commencé par Carnap. Il est plus juste de dire qu'il ont *accompli un travail différent.* Ils ont, en effet, prouvé que $S5$ quantifié est complet relativement à une autre interprétation de la nécessité que celle que visait Carnap. Ceci a été établi par N. Cocchiarella ([Cocchiarella 84], p. 312-317), et par F. Rivenc et Ph. de Rouilhan ([Rivenc et de Rouilhan 97], p. 32-33). Pour comprendre la nature et la portée du changement d'interprétation intervenu, il faut pénétrer plus avant dans la sémantique de la logique modale.

1.14. Carnap, précurseur de la sémantique des mondes possibles

On peut considérer Carnap comme le précurseur de la *sémantique des mondes possibles.* En effet, dans l'ouvrage *Meaning and Necessity*, paru en 1947 et 1956, Carnap élabore une sémantique qui précise le sens de notions modales telles que celle de vérité nécessaire et il le fait en s'inspirant de l'idée leibnizienne qu'une vérité nécessaire doit valoir dans tous les mondes possibles ([Carnap 47] traduction française par [Rivenc et de Rouilhan 97], p. 60). La notion de *monde possible* ainsi que la sémantique qui peut y être associée seront examinées en détail au chapitre 2.

Carnap s'engage dans un travail d'analyse logique, non dans une enquête empirique et descriptive. L'objectif qu'il vise est décrit avec beaucoup de netteté dans ce passage : "Rendre plus exact un concept vague ou pas tout à fait exact de la vie quotidienne ou d'une étape antérieure du développement scientifique ou logique, ou plutôt le remplacer par un concept plus exact construit à nouveaux frais, telle est l'une des tâches les plus importantes de l'analyse et de la construction logiques. Cette tâche, nous l'appelons *expliquer*, ou donner une *explication* de l'ancien concept; ce dernier, ou parfois le terme utilisé pour lui, est appelé l'*explicandum*, et le nouveau concept, ou le nouveau terme, *l'explicatum* de l'ancien" [Carnap 46], p. 56.

Prenant, par exemple, le concept leibnizien de vérité nécessaire comme *explicandum*, Carnap va l'expliquer à l'aide de la notion de "L-vrai" qui jouera le rôle d'*explicatum*. La notion de "L-vrai" (logiquement vrai) est une notion technique. Avant de la définir, il faut introduire deux notions préliminaires; la notion de *description d'état* et celle de *valoir dans une description d'état.*

Carnap définit ainsi la notion de description d'état : "Une classe d'énoncés du [système sémantique] $\mathcal{S}_1$ contenant, pour tout énoncé atomique, cet énoncé ou sa négation, mais pas les deux à la fois, et ne contenant aucun autre énoncé est appelée une *description d'état* dans $\mathcal{S}_1$, parce qu'elle fournit manifestement une description complète d'un état possible de l'univers des individus relativement à toutes les propriétés et relations exprimées par les prédicats du système" [Carnap 46], p. 58.

Carnap formule ainsi la définition de *valoir dans une description d'état* : "... dire qu'un énoncé vaut dans une description d'état, c'est dire qu'il serait vrai si

la description d'état (c'est-à-dire tous les énoncés qui lui appartiennent) était vraie"[Carnap 46], p. 58. Muni de ces deux définitions préliminaires, Carnap définit alors ce qu'il faut entendre par L-vrai.

Définition 1.1. Un énoncé $\mathbf{E}_j$ est L-vrai (dans $\mathcal{S}_1$) si $\mathbf{E}_j$ vaut dans toute description d'état (dans $\mathcal{S}_1$).

Dans la terminologie de Carnap, les lettres en gras sont des signes du métalangage; ainsi, $\mathbf{E}_j$ est mis pour les énoncés. Dire qu'un énoncé *vaut* dans une description d'état, c'est dire qu'il serait vrai si la description d'état était vraie [Rivenc et de Rouilhan 97], p. 58.

1.15. De la nécessité selon Carnap à la nécessité selon Kripke

Carnap interprète "*nécessairement p*" comme "*p* est une vérité de logique". Il définit la vérité de logique comme la vérité de p dans toutes les descriptions d'état. Si on reformule sa définition de la vérité logique dans la terminologie actuelle des mondes possibles, la nécessité, pour Carnap, se confond avec la validité.

Pour représenter formellement la validité, il faut faire appel à la logique du deuxième ordre. Or l'ensemble des formules valides de la logique du deuxième ordre n'est pas récursivement énumérable. On trouve une preuve concise de ce résultat dû à Gödel dans *A Mathematical Introduction to Logic* de Enderton (basée sur l'idée que "l'ensemble des nombres gödéliens des validités de second ordre n'est pas récursivement énumérable") [Enderton 72], p. 272-273.

Comme l'ensemble des théorèmes de la logique du deuxième ordre est récursivement énumérable, il s'ensuit que cet ensemble ne recouvre pas celui des formules valides. En d'autres termes, il y a des formules valides de la logique du deuxième ordre qui ne sont pas des théorèmes. La logique du deuxième ordre est incomplète et la logique modale, relativement à l'interprétation de la nécessité en termes de validité, l'est aussi. Néanmoins, Henkin a prouvé que la logique non modale du deuxième ordre, bien qu'elle soit incomplète pour la validité définie au sens usuel, redevient complète si on définit la validité comme étant la vérité dans tous les *modèles généraux* (ce qui constitue une restriction par rapport à la vérité dans tous les *modèles standard*). C'est de ce résultat que se sont servis Kripke ([Kripke 59], p. 2) et Bayart ([Bayart 59], p. 100) pour démontrer la complétude du système $S5$.

Rappelons qu'en logique du deuxième ordre, dans un modèle normal (appelé *modèle standard* par Henkin), le domaine des objets assignés aux variables prédicatives contient *tous* les sous-ensembles du domaine des individus (ou tous les sous-ensembles des produits cartésiens de ce domaine). On appelle *modèle général* un modèle où les domaines d'objets de variables prédicatives ne contiennent pas *tous* les sous-ensembles du domaine des individus (sur ces questions, voir [Hintikka 81].

Décrivant la démarche de Kripke – et cela s'applique aussi à Bayart qui se réfère explicitement au théorème de Henkin –, Cocchiarella écrit : "Comme pour la logique du deuxième ordre standard [non modale], l'incomplétude de la sémantique peut être évitée en permettant à l'interprétation quantificationnelle de la nécessité dans le métalangage de faire référence non à tous les mondes possibles (modèles) d'un espace logique donné, mais seulement à un ensemble donné non vide de tels mondes" [Cocchiarella 84], p. 315.

Pour prouver la complétude de $S5$ quantifié, Kripke a dû changer la conception de modèle et de monde possible implicite dans la sémantique de Carnap. Cette innovation ne se voit pas aisément car la sémantique de Carnap fait usage d'une autre terminologie que celle qu'utilisera Kripke. Carnap définit la vérité de logique en termes de descriptions d'état et non en termes de mondes possibles.

C'est pour cela que dans leur introduction à *Signification et nécessité* de Carnap, F. Rivenc et Ph. de Rouilhan ont reformulé la sémantique de Carnap dans la terminologie de Kripke, ce qui peut se faire sans dommage car "les descriptions d'état d'un langage carnapien (contenant une infinité de constantes d'individus désignant un à un les individus d'un domaine infini dénombrable) correspondent canoniquement aux mondes possibles sur ce domaine pour le langage obtenu en supprimant les constantes d'individus" [Rivenc et de Rouilhan 97], p. 32n.

Nous présenterons en détail cette reconstruction de la sémantique de Carnap en termes de mondes possibles dans le chapitre sur la théorie des modèles et la théorie de la preuve pour la logique modale quantifiée. Ce que nous avons dit ici de cette reconstruction suffit pour caractériser l'innovation qui a permis à Kripke de prouver la complétude de $S5$ quantifié.

Kripke introduit un concept de monde possible différent de celui qui correspond à la description d'état chez Carnap. F. Rivenc et Ph. de Rouilhan caractérisent ainsi l'innovation : "La notion de modèle et, corrélativement, l'interprétation de l'opérateur de nécessité, ne font plus référence à tous les mondes du domaine donné, mais seulement à une partie quelconque non vide d'entre eux" [Rivenc et de Ph. Rouilhan 97], p. 33. (De même, Henkin ne faisait plus appel à tous les modèles, mais seulement à une partie d'entre eux.)

Avec l'introduction de ce nouveau concept de monde possible, la notion de validité qui repose sur lui change aussi et le concept de nécessité que la validité est censée expliquer change. La nécessité dont il est question dans l'article de Kripke de 1959, F. Rivenc et Ph. de Rouilhan l'appellent *nécessité métaphysique* pour la distinguer de la *nécessité logique* expliquée par Carnap dans *Signification et Nécessité*. Le terme de *nécessité métaphysique* a été employé par Kripke dans *Identity and Necessity* [Kripke 71], p. 1.

1.16. Domaines variables et relation d'accessibilité

En 1963, Kripke innove sur deux points :

1. Il associe des domaines différents aux mondes d'un même modèle, c'est-à-dire des *ontologies locales* pour reprendre l'expression de N. Lavand (Hintikka, traduction et introduction de [Lavand 89]).
2. Il introduit la relation d'accessibilité entre mondes possibles. Un monde n'est plus possible absolument, mais relativement à un autre. Pour les systèmes axiomatiques de logique modale les plus courants, on peut établir une *correspondance* entre axiomes modaux et propriétés de la relation d'accessibilité, ce qui facilite grandement la comparaison entre systèmes ([Sahlqvist 75], [van Benthem 76], et § 2.2.5).

Prior, dans un texte inédit, définit déjà en 1956 les opérateurs modaux (qu'il écrit L pour "nécessairement" et M pour "possiblement") à l'aide de quantificateurs et d'une relation binaire U :

1. $(Lp)a =_{def} \forall b(Uab \supset pb)$;
2. $(Mp)a =_{def} \exists b(Uab \wedge pb)$.

Ces deux relations peuvent respectivement se lire :

1. a vérifie nécessairement p si et seulement si, pour tout b, si a est dans la relation U à b, alors b vérifie p.
2. a vérifie possiblement p si et seulement si, pour quelque b, a est dans la relation U à b et b vérifie p.

Prior et Meredith n'ont pas paraphrasé la relation U en langue naturelle. On voit tout de suite que si l'on assigne aux variables a et b un domaine de mondes possibles et que l'on interprète Uab comme "le monde b est accessible à partir du monde a", on obtient la sémantique de Kripke. Dans ce cas, la relation U est la *relation d'accessibilité*, que l'on notera R (voir le chapitre 2). Prior et Meredith n'ont pas franchi le pas de cette interprétation.

Prior et Meredith énoncent des axiomes exprimant respectivement la réflexivité, la symétrie et la transitivité de U et associent à ces axiomes les propriétés de la relation d'accessibilité qui correspondent effectivement à ces propriétés, mais Prior et Meredith restent centrés sur la théorie de la preuve. En 1960, Geach proposa à Prior de donner à U le nom de "relation d'accessiblité". La relation binaire d'accessibilité a ensuite été réinventée par Kanger en 1957, par Hintikka, la même année, et par Kripke en 1959 ([Copeland 96], p. 11-13) et intégrée cette fois à la sémantique des systèmes formels.

Dans le cadre de la sémantique des mondes possibles, la relation d'accessibilité sert à donner les *conditions de vérité* des formules modales, c'est-à-dire leur sens :

> "Possiblement A" est vrai dans le monde w relativement au modèle $\mathcal{M}$ si et seulement s'il existe un monde possible w' tel que A est vrai dans w', et w' est accessible à partir de w.

Malgré les apparences, cette clause n'est nullement circulaire. Elle ne se présente pas comme une définition de "possiblement", mais comme une clause

de la définition récursive de la *vérité* pour un langage modal, définition qui donne les conditions de vérité d'une formule complexe à partir des conditions de vérité de ses composants (ou des conditions de satisfaction des fonctions propositionnelles). Le terme "monde possible" est pris comme notion primitive. Pour le logicien dont l'objectif est de donner une preuve de complétude, "il n'est pas nécessaire d'analyser davantage la notion de *monde possible*" [Kripke 59], p. 2.

Le rôle de la relation d'accessibilité est précisé par Hintikka dans *Grammar and Logic : Some Borderline Problems* : "Pour dire les choses brièvement et même crûment, en marchant d'un monde vers ses alternatives, nous pouvons réduire les conditions de vérité des énoncés modaux aux conditions de vérité d'énoncés non modaux" [Hintikka 73], p. 198.

Considérant dans son livre *L'objet quelconque* l'autre innovation de Kripke, à savoir le fait d'associer des domaines d'individus différents aux divers mondes, F. Nef souligne que cette régionalisation des ontologies "crée un certain nombre de problèmes métaphysiques qui ont fait à la fois la fortune des modèles de Kripke – en permettant de réinterpréter la métaphysique traditionnelle de l'essence et de l'identité – mais aussi [qu'elle] conduit à chercher des solutions alternatives" [Nef 98], p. 258.

A noter enfin, que la relation d'accessibilité (U) est un concept qui sera amplement développé, puis utilisé, dans les chapitres qui suivent; cette relation y sera notée wRw' pour indiquer que le monde w est en relation avec le monde w', c'est-à-dire que w' est accessible à partir de w.

1.17. De l'implication stricte à l'implication relevante

Le recours à l'implication stricte permet de se débarrasser du principe *ex falso sequitur quodlibet*, mais non du principe selon lequel "d'une contradiction découle n'importe quoi". Le problème n'a donc pas été résolu, mais seulement déplacé.

Cependant, C. Lewis assume ces conséquences indésirables de son nouveau connecteur et voit dans le principe "*ex contradictione sequitur quodlibet*" un fait à propos de la relation de déductibilité [Lewis and Langford 32], p. 251.

L'argument qu'il présente a été reconstruit de la manière suivante par A. Anderson et N. Belnap en 1968. Les règles utilisées dans le tableau suivant sont celles de la déduction naturelle (I, p. 138).

(a)	$A \wedge \neg A$	contradiction
(b)	A	de (a) par simplification
(c)	$A \vee B$	de (b) par addition
(d)	$\neg A$	de (a) par simplification
(e)	B	de (a) et (d) par syllogisme disjonctif

Ce raisonnement est contesté par Anderson et Belnap. Dans ce raisonnement le connecteur $\vee$, la disjonction, est utilisé dans deux sens; un *sens extensionnel*

(à l'œuvre dans la règle de l'addition autorisant le passage de (b) à (c)) et un *sens intensionnel* (à l'œuvre dans le syllogisme disjonctif : $A \vee B$, $\neg A$ donc B). Ce dernier n'est valide que s'il y a une relation intensionnelle entre A et B [Anderson and Belnap 68], p. 106-108.

Dans *Les systèmes formalisés des modalités aristotéliciennes*, R. Feys précise en ces termes l'idéal visé par les logiciens qui cherchent une implication qui coïnciderait avec la déductibilité. Il faudrait "en faire une relation *intensive*, un *lien d'idées*, et pas une relation *extensionnelle*" [Feys 50], p. 482.

On voit tout de suite que l'implication stricte $-<$, bien qu'elle ne soit pas extensionnelle, ne peut satisfaire cette exigence. En effet le composant intensionnel de $p-< q$ (qui est défini par $\Box(p \supset q)$), c'est l'opérateur unaire $\Box$ qui exprime une *propriété* (la propriété de nécessité attribuée à la proposition "si p alors q" prise comme un tout). Ce qu'on voudrait, c'est une *relation intensionnelle* entre p et q. Le travail a été repris par A. Church dans *The weak theory of Implications* [Church 51], par W. Ackermann dans *Begründung einer strengen Implikation* [Ackermann 56], et par A. Anderson et N. Belnap dans *The pure Calculus of Entailment* [Anderson and Belnap 62].

Dès 1928, I. Orlov avait axiomatisé un fragment de ce qui sera la logique relevante d'Anderson et Belnap, fragment comprenant le conditionnel relevant et la négation [Došen 92]. L'implication relevante du système $R'_{\rightarrow}$, implication notée $\rightarrow$, est *formellement définie* par le théorème suivant [Anderson and Belnap 75], p. 20 :

> $A \rightarrow B$ est un théorème de $R'_{\rightarrow}$ dans le cas où il existe une preuve de B *à partir* l'hypothèse de A, dans laquelle A a effectivement servi.

Il reste alors à énumérer les quatre axiomes du fragment implicationnel de $R'_{\rightarrow}$, c'est-à-dire de la logique relevante qui a pour seul connecteur l'implication, et à décrire un procédé de marquage des prémisses effectivement utilisées.

L'implication relevante d'Anderson et Belnap capte bien l'idée d'une relation de pertinence entre l'antécédent et le conséquent, mais le système axiomatique qui définit le conditionnel correspondant était dépourvu d'une sémantique satisfaisante. Un progrès décisif a été réalisé par G. Priest et R. Sylvan dans leur article *Simplified Semantics for Basic Relevant Logic* [Priest and Sylvan 92]. Leur sémantique fait usage des outils conceptuels de la sémantique des mondes possibles. Comme on a besoin d'une *relation binaire* d'accessibilité pour capter le sens de l'opérateur unaire "nécessairement", on ne s'étonnera pas qu'il faille recourir à une *relation d'accessibilité ternaire* pour capter le sens du conditionnel relevant qui est un connecteur binaire. (Concernant la théorie de la preuve automatisée en logique relevante, voir [Bloesch 93], [P. Gochet et al. 95] et [Riche 91].)

Dès que l'on dispose de l'implication relevante on peut aisément définir la disjonction relevante $\underline{\vee}$: $p \underline{\vee} q =_{def} \neg p \rightarrow q$.

1.18. La diversification de la logique modale

La logique modale s'est diversifiée en plusieurs secteurs. Le premier traitement formel de la *logique déontique* remonte à un élève de Meinong, Ernst Mally, qui publia en 1926 *Grundgesetze des Sollens. Elemente der Logik des Willens.* En 1951 paraît dans la revue *Mind* le texte fondateur de H. von Wright "Deontic Logic" [von Wright 51]. La même année, G. Kalinowski défend, à Lublin, sa thèse sur *La logique des propositions pratiques.* Son livre, *La Logique déductive*, publié en 1996 comporte un chapitre sur la logique déontique, un chapitre sur la logique des normes et s'achève sur des considérations sur l'utilisation de la logique déductive dans le domaine du droit [Kalinowski 96].

L'ouvrage fondateur de la *logique temporelle* est *Time and Modality* d'A. Prior [Prior 57]. En langue française, l'ouvrage de J. Gardies reste la principale référence [Gardies 75].

La *logique épistémique* commence avec *Knowledge and Belief* [Hintikka 62]. En 1995, ont paru deux ouvrages sur le sujet : *Reasoning About Knowledge* de R. Fagin, J. Halpern,Y. Moses et M. Vardi [Fagin et al. 95], ainsi que *Epistemic Logic for A.I. and Computer Science* de J.-J. Meyer et W. van der Hoek [Meyer and van der Hoek 95].

La *logique des conditionnels contrefactuels* est très importante pour la philosophie des sciences. Plusieurs philosophes – parmi lesquels Ch. Peirce – rendent compte de la distinction entre les énoncés de loi et les généralisations accidentelles par la présence implicite de conditionnels contrefactuels dans les énoncés de loi. Ces derniers disent "ce qui arriverait si...". Parmi les textes fondateurs de la logique des conditionnels, on peut citer *A Theory of Conditionals* [Stalnaker 68], et *Counterfactuals* [Lewis 73]. Le livre de G. Crocco, *Fondements logiques du raisonnement contextuel*, étudie tous les aspects de la logique des conditionnels [Crocco 96].

La preuve de complétude de la *logique dynamique* remonte à l'article de Parikh [Parikh 78]. Le sujet a été repris par Harel dans *Dynamic Logic* publié dans le *Handbook of Philosophical Logic* [Harel 84]. Nous avons étudié le rapport entre la sémantique dynamique et le système de Lewis dans *Quantified Modal Logic, Dynamic Semantics and S5* [Gochet and Gillet].

La jonction entre la logique modale et la logique trivalente fait l'objet de nombreuses études dans les *Essais sur les logiques non chrysipiennes* [Moisil 72] et récemment dans *Eléments de logique trivalente* de [Ntambue 98].

Chapitre 2
Logique modale des propositions

2.1. Introduction

Dans le cadre d'une langue naturelle, les *modalités* sont des expressions qui modifient le sens d'une autre expression. Par exemple, un adverbe de phrase modifie le sens d'une phrase tandis qu'un adjectif sert généralement à modifier celui d'un nom commun. La *logique modale* est la langue de la logique qui formalise les modalités de la langue naturelle. La logique modale n'est pas une langue unique, mais un ensemble de langues, dont chacune traduit certaines des modalités de la langue naturelle. Plutôt que de "*la* logique modale", il faudra donc parler "*des* logiques modales".

Par exemple, la *logique aléthique* formalise les modalités du "possible" et du "nécessaire", la *logique déontique* celles du "permis", de l'"obligatoire" et de l'"interdit", la *logique épistémique* traite des modes de "connaissance" et de "croyance", tandis que la *logique temporelle* introduit les modes "parfois" et "toujours" (dans le "futur" et dans le "passé"), ainsi que leurs antonymes "souvent" et "jamais".

En vue de modéliser les modalités de la langue naturelle, on étendra le langage formel des logiques classiques en y ajoutant des opérateurs unaires qui portent sur des constantes de ce langage. La logique modale la plus élémentaire est celle dotée d'un opérateur, noté $\Box$, qui formalise la "nécessité", et qui porte sur des formules pour en modifier le sens. Comme les quantificateurs de la logique prédicative, les opérateurs modaux sont introduits par paires et sont reliés entre eux par une relation de dualité. Ce sont, en logique aléthique, les opérateurs modaux *universel* (noté $\Box$) et *existentiel* (noté $\Diamond$), et qui représentent respectivement les modes du "nécessaire" et du "possible".

Les langues naturelles sont des *langues multimodales* en ce sens que diverses modalités y sont présentes. La phrase "Marie croit que Jean sera obligé de travailler" contient les modalités déontique ("obligé de"), temporelle ("sera") et épistémique ("croit que"). Les logiques modales devront donc être conçues de façon à pouvoir représenter formellement des fragments de la langue naturelle comprenant plusieurs types de modalités.

Donnons quelques exemples de logiques multimodales couramment utilisées en intelligence artificielle. Les logiques épistémiques qui formalisent les croyances ou les connaissances de plusieurs agents sont des langues multimodales. Les logiques temporelles sont naturellement multimodales, le futur et le passé étant traités comme deux modalités distinctes (quoique non indépendantes). Il est par ailleurs courant de devoir formaliser à la fois les modalités temporelle et aléthique dans le cadre d'un langage multimodal approprié; la logique intensionnelle de Montague en est un exemple. La logique dynamique qui modélise les commandes d'un langage de programmation comporte autant de modalités qu'il y a de commandes différentes.

Les langues formelles *multimodales* sont caractérisées par des paires d'opérateurs portant sur des constantes (généralement des formules) pour en modifier le sens; ces opérateurs sont notés $[i]$ et $<i>$.

Les logiques modales sont formées au moyen des langues de la logique classique auxquelles on a ajouté une ou plusieurs composantes modales. Les logiques modales peuvent donc aussi être classifiées selon les langues logiques classiques qui en constituent le noyau. Les logiques les plus courantes étant la *logique des propositions* et la *logique des prédicats*, on distinguera donc également les *logiques modales des propositions* des *logiques modales des prédicats*.

On peut faire remonter les débuts de la logique modale à l'antiquité grecque. Aristote, les stoïciens et les mégariens ont introduit les rudiments d'une logique modale temporelle. La logique modale moderne débute avec Lewis qui fit apparaître les limites de la logique classique dans le cadre de la modélisation de l'implication ([Lewis 12]; voir aussi [Gabbay and Guenthner 84]). L'approche préconisée par Lewis est purement syntaxique et on n'y trouve donc pas trace de sémantique. Une deuxième approche de la logique modale est basée sur la sémantique des logiques à plusieurs valeurs (ou *sémantique algébrique*) dont un des principaux inventeurs est Łukasiewicz ([Łukasiewicz 53, 70]; voir aussi [Gabbay and Guenthner 84]). Une troisième approche est celle du formalisme sémantique faisant usage de la *théorie des modèles*. Il semble bien que cette approche soit actuellement la plus prometteuse. La *sémantique des mondes possibles* est la forme la plus usuelle des sémantiques basées sur la théorie du modèle; elle est présentée dans la section 2.2. La sémantique algébrique est introduite dans la section 2.3.

2.2. Logique modale des propositions

2.2.1. *Introduction*

Nous percevons intuitivement la différence qualitative existant entre une proposition *vraie mais contingente*, telle que : "La marquise sortit à cinq heures", et des propositions *nécessairement vraies*, telles que : "Sept et cinq font douze" (nécessité mathématique), "Le granit n'est pas inflammable" (nécessité physique), "Si Jean est célibataire, alors il n'est pas marié" (nécessité de langage). Ces différentes sortes de *vérités* ne sont pas représentables en logique classique; il faut donc inventer une langue formelle plus puissante pour pouvoir les analyser.

Les énoncés suivants suggèrent un début de formalisation :

- Une affirmation est *nécessairement vraie* si elle est vraie dans toutes les situations imaginées; non seulement dans *notre* situation, mais également dans tout un ensemble, éventuellement infini, de situations imaginables.
- Une affirmation est *contingentement vraie* si elle est vraie dans certaines circonstances bien précises; par exemple un jour où il ne pleut pas, et là où nous nous trouvons.

Il s'agit ensuite de préciser ce que l'on entend par ensemble de circonstances, de situations imaginables, réelles, possibles, etc. Plutôt que de situation ou de circonstance, et pour se conformer au vocabulaire en usage dans la *sémantique des mondes possibles*, on utilisera dorénavant le terme de *monde*. L'ensemble des mondes, ou *univers*, qu'il faut considérer est fonction des différentes sortes de nécessités; par exemple :

- **Nécessité logique :** Ensemble des mondes dans lesquels les lois de la logique sont les mêmes que celles que nous connaissons.
- **Nécessité physique :** Ensemble des mondes dans lesquels les lois de la physique sont les mêmes que celles que nous connaissons dans "notre monde"; par exemple des mondes dans lesquels la température absolue est supérieure à -273 degrés et où les objets se déplacent à une vitesse inférieure à celle de la lumière.

Une autre limitation de la logique classique provient de la définition de ses connecteurs, et, en particulier, de celle de l'implication matérielle. Selon cette définition, l'assertion "si A alors B" est vraie dans deux cas :

1. Quand A et B sont tous deux vrais.
2. Quand A est faux indépendamment de B.

L'implication matérielle ne peut être utilisée pour caractériser les objets au moyen de leurs propriétés, ce qui est une façon de faire classique en physique. Par exemple, la qualification de "magnétique", appliquée à un objet, est opérationnellement bien définie par l'assertion :

Un objet est magnétique *si*, des brins de limaille de fer se trouvant dans son voisinage, *alors* ceux-ci sont attirés par cet objet.

Selon l'implication matérielle, un objet au voisinage duquel ne se trouveraient pas des brins de limaille de fer serait qualifié de "magnétique", ce qui est évidement incorrect.

2.2.2. *Syntaxe de la logique modale*

Selon une formulation attribuée à Leibniz, un énoncé est *nécessairement vrai* s'il est vérifié dans tous les mondes possibles tandis qu'un énoncé est *possiblement vrai* s'il est vérifié dans au moins un monde possible.

Pour formaliser ces notions de *nécessairement vrai* et de *possiblement vrai* le langage modal comportera donc deux symboles supplémentaires (par rapport à ceux du langage logique de base qu'il généralise) : $\Box$ et $\Diamond$, qui sont appelés respectivement *opérateur modal universel* et *opérateur modal existentiel*. Le langage ainsi créé est une logique modale aléthique; sa syntaxe (qui est également celle des autres logiques modales) est définie par les deux règles suivantes :

1. Toutes les règles syntaxiques de la logique des propositions sont aussi des règles syntaxiques de la logique modale.
2. Si A est une formule, alors $\Box A$ et $\Diamond A$ sont des formules.

Dans le cadre d'une logique de nécessité, l'opérateur $\Box$ est interprété comme "il est nécessaire que", tandis que l'opérateur $\Diamond$ est interprété comme "il est possible que". On impose à ces opérateurs une contrainte de dualité :

$$\Diamond A =_{def} \neg \Box \neg A. \tag{2.1}$$

Cette contrainte est dictée par la relation qui lie, en langue naturelle, les notions de "nécessité" et de "possibilité", à savoir que "si quelque chose est possible, il est faux que son contraire soit nécessaire". Dans le cadre de la langue logique, on peut considérer cette relation comme une définition de l'opérateur $\Diamond$ au moyen de l'opérateur $\Box$, si bien que les formules modales peuvent toujours s'écrire au moyen du seul opérateur $\Box$.

On observe l'analogie entre cette contrainte de dualité et celle qui lie les quantificateurs universel et existentiel en logique du premier ordre :

$$\exists x A =_{def} \neg \forall x \neg A. \tag{2.2}$$

La logique modale utilise les opérateurs $\Box$ et $\Diamond$ sans leur attribuer une interprétation *a priori* en langue naturelle. C'est dire que d'autres interprétations que celles de nécessité et de possibilité peuvent être associées à ces opérateurs, pourvu que ces interprétations satisfassent la relation de dualité (2.1), encore appelée *antonymie*.

La correspondance existant entre certaines lectures pour les formules $\Box A$ et $\Diamond A$ est illustrée au moyen du tableau de la figure 2.1.

LECTURES POUR $\Box A$	LECTURES POUR $\Diamond A$
Il est nécessaire que A	Il est possible que A
Il sera toujours vrai que A	Il sera parfois vrai que A
Il faut que A	Il est permis que A
Il est su que A	L'inverse de A n'est pas su
Il est connu que A	L'inverse de A n'est pas connu
Toute exécution du programme produit le résultat A	Il y a une exécution du programme qui produit le résultat A

Figure 2.1 *Lectures pour les opérateurs modaux*

2.2.3. *Sémantique de la logique modale*

La sémantique de la logique modale est construite à partir des concepts de structure et de modèle.

Une *structure* est un couple $\mathcal{F} = < W, R >$, où W est un ensemble non vide et où R est une relation binaire sur l'ensemble W, c'est-à-dire un sous-ensemble de $W \times W$. L'ensemble W est appelé "univers" et les éléments de W sont les "mondes possibles". Dans ce cadre, la relation binaire R est appelée "relation d'accessibilité". Une structure est donc constituée d'un ensemble W de mondes possibles liés par une relation d'accessibilité R. Soient deux mondes s et t appartenant à l'univers W; le fait que le monde t soit accessible à partir du monde s se note sRt.

Soit P l'ensemble des atomes (ou propositions) d'un langage modal $\mathcal{L}$. Un *modèle* sur une structure $< W, R >$ est un triplet $\mathcal{M} = < W, R, V >$, où V est une application de P dans 2^W, qui associe à chaque proposition p de P un sous-ensemble $V(p)$ de W. On interprète $V(p)$ comme l'ensemble des points de W où la proposition p est "vraie".

Soit w un élément de W et soit A une formule modale. La sémantique des formules de $\mathcal{L}$ au point w est décrite au moyen de la notation :

$$\mathcal{M} \models_w A,$$

qui signifie "A est vrai au monde w dans le modèle $\mathcal{M}$". Les formules de la langue propositionnelle pouvant être écrites au moyen des propositions atomiques, de la constante $\perp$ (le *faux*), et du connecteur d'implication $\supset$, les formules de la langue modale requérant en plus l'opérateur universel $\Box$, la sémantique de la logique modale propositionnelle est entièrement définie au moyen des règles suivantes :

$\mathcal{M} \not\models_w \bot$;
$\mathcal{M} \models_w p$ ssi $w \in V(p)$;
$\mathcal{M} \models_w A_1 \supset A_2$ ssi $\mathcal{M} \models_w A_1$ implique $\mathcal{M} \models_w A_2$;
$\mathcal{M} \models_w \Box A$ ssi wRt implique $\mathcal{M} \models_t A$ pour tout $t \in W$.

La dernière règle exprime que la formule $\Box A$ est vraie au monde w d'un modèle $\mathcal{M}$ si la formule A est vraie en tous les mondes t qui sont en relation R avec le monde w. Ces définitions constituent les relations sémantiques de base. L'évaluation sémantique des formules $\top$ (le *vrai*), $\neg A$, $\Diamond A$, $(A_1 \vee A_2)$, $(A_1 \wedge A_2)$ et $(A_1 \equiv A_2)$ est obtenue à partir des relations sémantiques de base et des formules de transformation permettant de récrire la constante $\top$, l'opérateur $\Diamond$ et les connecteurs $\vee$, $\wedge$, $\equiv$ au moyen de la constante $\bot$, de l'opérateur $\Box$ et des connecteurs $\neg$ et $\supset$. Nous déduisons par exemple des relations $\neg A =_{def} (A \supset \bot)$ et (2.1) les règles sémantiques suivantes :

- $\mathcal{M} \models_w \neg A$ ssi $\mathcal{M} \not\models_w A$;
- $\mathcal{M} \models_w \Diamond A$ ssi $\mathcal{M} \models_t A$ pour au moins un $t \in W$ tel que wRt.

Ayant défini la valeur de vérité d'une formule modale au moyen des règles sémantiques, on peut introduire les notions de formule *valide dans un modèle*, de formule *valide dans une structure* et de formule *valide*.

- Une formule A est *valide dans un modèle* $\mathcal{M}$ si elle est vraie en tous les mondes du modèle, c'est-à-dire si $\mathcal{M} \models_w A$ pour tout $w \in W$; ceci se note $\mathcal{M} \models A$.
- Une formule A est *valide dans une structure* $\mathcal{F} = <W, R>$ si elle est valide dans tout modèle possédant cette structure, c'est-à-dire si $\mathcal{M} \models A$ pour tout modèle $\mathcal{M}$ constitué d'une valuation (quelconque) et de la structure $\mathcal{F}$; ceci se note $\mathcal{F} \models A$.
- Une formule A est *valide* si elle est valide dans toute structure $\mathcal{F}$; ceci se note $\models A$.

2.2.4. *Interprétations de la logique modale*

Nous avons indiqué à la figure 2.1 diverses lectures possibles pour les formules modales $\Box A$ et $\Diamond A$. Ces lectures vont donner lieu à différentes logiques modales.

2.2.4.1. *Logique du possible et du nécessaire*

Dans ce contexte on suppose que la relation d'accessibilité R est *réflexive*, c'est-à-dire que l'on a sRs pour tout $s \in W$, où encore que tout monde est accessible à partir de lui-même.

Une *vérité nécessaire* dans le monde s est une formule qui est vérifiée dans tous les mondes t accessibles à partir de s; la formule $\Box A$ se lit "A est nécessairement vrai".

Une *vérité possible* dans le monde s est une formule qui est vérifiée dans au moins un monde accessible à partir de s; la formule $\Diamond A$ se lit "il est possible que A soit vrai".

Il est souvent utile de particulariser un des mondes de l'univers comme le "*monde réel*". Une formule A qui est nécessairement vraie dans le monde réel sera vraie dans tous les mondes accessibles à partir du monde réel et donc en particulier dans le monde réel lui-même.

La formule $\Box A \supset A$ est valide dans la structure qui modélise l'assertion selon laquelle une "vérité nécessaire dans un monde" est celle qui est vérifiée dans tous les mondes accessibles à partir de ce monde, y compris dans le monde lui-même. Si la formule A est vraie dans le monde réel, elle le sera par conséquent dans au moins un monde accessible à partir de ce monde; la formule $A \supset \Diamond A$ est donc également valide dans la structure qui modélise l'assertion précitée.

2.2.4.2. *Logique temporelle*

En logique temporelle, les mondes possibles représentent les états du monde évalués à différents moments. Le monde réel est l'état du monde à un certain instant initial pris comme instant de référence. La relation d'accessibilité sRt indique dans ce cas que "t est postérieur à s". La formule $\Box A$ se lit "à tous les instants futurs A (sera vrai)" tandis que la formule $\Diamond A$ se lit "à au moins un instant futur A (sera vrai)". Si la relation d'accessibilité sRt indique que "t est antérieur à s", les formules $\Box A$ et $\Diamond A$ se lisent respectivement "à tous les instants passés A (était vrai)" et "à au moins un instant passé A (était vrai)".

Des structures $<W, R>$ qui apparaissent comme naturelles pour la logique temporelle sont celles où W est un des ensembles $\mathbf{N}$ (nombres naturels), $\mathbf{Z}$ (nombres entiers), $\mathbf{Q}$ (nombres rationnels), $\mathbf{R}$ (nombres réels), et où R est une des relations de précédence $<$, $\leq$, $>$, $\geq$.

L'image classique du temps est celui utilisé en physique newtonienne. Le temps y est un "continuum linéaire à une dimension". On est aussi conduit à utiliser dans certaines théories physiques un modèle de temps non linéaire, où un moment peut, par exemple, avoir plusieurs futurs non reliés entre eux par la relation d'accessibilité.

2.2.4.3. *Logique dynamique*

La logique dynamique est basée sur l'association d'un opérateur modal à chaque commande d'un langage de programmation. Dans ce contexte, l'ensemble W est interprété comme l'ensemble des états possibles au cours d'un calcul. La propriété sRt indique qu'un calcul débute dans l'état s et se termine dans l'état t. Un programme non déterministe peut conduire à plusieurs résultats. La formule $\Box A$ signifie alors "toutes les exécutions du programme se terminent avec la formule A vraie". La formule $\Diamond A$ signifie "au moins une exécution du programme se termine avec la formule A vraie".

2.2.4.4. *Logiques de croyance et de connaissance*

Si la fonction première de la logique modale est la formalisation des modes de "nécessité" et de "possibilité", un de ses autres champs d'application préférentiels est la modélisation et l'analyse des paradigmes de "connaissance" et de "croyance". A cette fin, différents systèmes logiques utilisent des langages formels comportant des opérateurs de "croyance" et de "connaissance". Dans le cadre des logiques de croyance et de connaissance, l'opérateur $\Box$ prend respectivement la signification "est cru" et "est connu". Son dual, l'opérateur $\Diamond$, prend respectivement la signification "l'inverse n'est pas cru" et "l'inverse n'est pas connu".

2.2.4.5. *Formules remarquables*

Différentes formules particulières (ou plutôt schémas de formules) jouent un rôle prépondérant en logique modale. La plupart de ces formules ont reçu un nom au cours de l'histoire de la logique modale; quelques-unes d'entre elles sont rassemblées ci-dessous, en regard de leur appellation historique.

K : $\Box(A \supset B) \supset (\Box A \supset \Box B)$
P : $A \supset \Box A$
L : $\Box(\Box A \supset A) \supset \Box A$
M : $\Box\Diamond A \supset \Diamond\Box A$

D'autres schémas remarquables sont présentés dans le paragraphe suivant; ils recevront le statut de *schéma associé à une propriété de la relation d'accessibilité.*

2.2.5. *Relations binaires et schémas de formules*

La liste suivante est celle de propriétés intéressantes que peut posséder une relation binaire R.

1.	Réflexive	:	$\forall s\,(sRs)$
2.	Symétrique	:	$\forall s \forall t\,(sRt \supset tRs)$
3.	Reproductive (ou sérielle)	:	$\forall s \exists t\,(sRt)$
4.	Transitive	:	$\forall s \forall t \forall u\,(sRt \wedge tRu \supset sRu)$
5.	Euclidienne	:	$\forall s \forall t \forall u\,(sRt \wedge sRu \supset tRu)$
6.	Partiellement fonctionnelle	:	$\forall s \forall t \forall u\,(sRt \wedge sRu \supset t = u)$
7.	Fonctionnelle	:	$\forall s \exists! t\,(sRt)$
8.	Faiblement dense	:	$\forall s \forall t\,(sRt \supset \exists u\,(sRu \wedge uRt))$
9.	Faiblement connexe	:	$\forall s \forall t \forall u\,(sRt \wedge sRu \supset tRu \vee t = u \vee uRt)$
10.	Faiblement dirigée	:	$\forall s \forall t \forall u(sRt \wedge sRu \supset \exists v\,(tRv \wedge uRv))$

(La notation $\exists! t$ signifie : il existe un et un seul t.)

A cette liste de propriétés correspond une liste de schémas de formules.

1.	$\Box A \supset A$	(T)
2.	$A \supset \Box\Diamond A$	(B)
3.	$\Box A \supset \Diamond A$	(D)
4.	$\Box A \supset \Box\Box A$	(4)
5.	$\Diamond A \supset \Box\Diamond A$	(5)
6.	$\Diamond A \supset \Box A$	(Q)
7.	$\Diamond A \equiv \Box A$	
8.	$\Box\Box A \supset \Box A$	(R)
9.	$\Box(A \wedge \Box B \supset B) \vee \Box(B \wedge \Box B \supset A)$	
10.	$\Diamond\Box A \supset \Box\Diamond A$	(G)

Certains de ces schémas de formules sont repris comme schémas d'axiomes dans des systèmes axiomatiques. Les noms historiques attribués à ces schémas d'axiomes sont indiqués entre parenthèses.

Le théorème suivant, démontré dans [Goldblatt 87], énonce de manière précise la correspondance évoquée plus haut entre propriétés de la relation d'accessibilité et schémas valides dans les structures possédant ces propriétés.

Théorème 2.1. Etant donné une structure $\mathcal{F} = <W, R>$, la relation R possède une des propriétés 1–10 si et seulement si le schéma correspondant est valide dans $\mathcal{F}$.

Nous donnons, à titre d'exemple, la démonstration pour une des propriétés ci-dessus.

Hypothèse. La relation R est transitive, c'est-à-dire :

$$\forall s \forall t \forall u \, (sRt \wedge tRu \supset sRu).$$

Thèse. Soit $\mathcal{M}$ un modèle sur cette structure R; alors :

$$\mathcal{M} \models \Box A \supset \Box\Box A.$$

Démonstration. Soit s un monde tel que $\mathcal{M} \models_s \Box A$; il faut prouver que $\mathcal{M} \models_s \Box\Box A$. Nous avons successivement :

$\mathcal{M} \models_s \Box\Box A \iff (sRt \Longrightarrow \mathcal{M} \models_t \Box A) \iff (sRt \Longrightarrow (tRu \Longrightarrow \mathcal{M} \models_u A))$
Si sRt et tRu, par transitivité sRu; nous avons donc par : $\mathcal{M} \models_s \Box A \Longrightarrow \mathcal{M} \models_u A$, ce qui démontre la thèse.

Le théorème 2.1 est fondamental; il explique le succès rencontré par la sémantique relationnelle depuis son introduction par [Kripke 63]. Les structures sont faciles à utiliser; en outre de nombreux schémas modaux correspondent aux propriétés importantes de la relation R. Il faut cependant noter que certaines propriétés usuelles d'une relation binaire R ne correspondent pas à la validité d'un schéma modal; telles sont par exemple les trois propriétés suivantes.

Non-réflexivité	:	$\forall s \neg (sRs)$
Antisymétrie	:	$\forall s \forall t \, (sRt \wedge tRs \supset s = t)$
Asymétrie	:	$\forall s \forall t \, (sRt \supset \neg(tRs))$

On a un moment espéré que le théorème 2.1 aurait pu servir de pierre angulaire dans la démonstration de l'équivalence entre la logique modale propositionnelle et la logique du premier ordre. Ces deux logiques ne sont

en fait pas équivalentes. En effet, on peut par exemple montrer que le schéma de formule M : $\Box\Diamond A \supset \Diamond\Box A$ ne peut être défini par un ensemble de formules du premier ordre [Goldblatt 87].

2.2.6. *Systèmes axiomatiques*

2.2.6.1. *Valuations et tautologies*

Une formule A est *quasi-atomique* soit si elle est atomique ($A \in P$, ensemble des atomes), soit si elle commence par le symbole $\Box$, c'est-à-dire si $A = \Box B$ où B est une formule atomique.

Soit P^q l'ensemble de toutes les formules quasi-atomiques; toute formule A peut être construite au moyen d'éléments de $P^q \cup \{\bot\}$ et du connecteur $\supset$. Il s'ensuit que la définition sémantique relative à ce connecteur ainsi que les valuations des formules quasi-atomiques :

$$V_w : P^q \to \{\mathbf{V} \ (vrai), \mathbf{F} \ (faux)\}, w \in W,$$

permettent d'étendre les valuations V_w à toute formule A du langage modal $\mathcal{L}$ engendré par P. On obtient ainsi les valuations :

$$V_w : \mathcal{L} \to \{\mathbf{V}, \mathbf{F}\}.$$

Une formule $A \in \mathcal{L}$ est une *tautologie* si $V_w(A) = \mathbf{V}$ pour toute valuation V_w de ses sous-formules quasi-atomiques.

2.2.6.2. *Logiques*

Etant donné un langage $\mathcal{L}$ basé sur un ensemble P d'atomes, on définit une *logique* (ou *théorie*) Λ comme un sous-ensemble de formules engendrées à partir de P et satisfaisant aux conditions suivantes :

- Λ contient toutes les tautologies;
- si A et $A \supset B$ sont des éléments de Λ, alors B est aussi un élément de Λ;
- si A est un élément de Λ, alors toute formule A' obtenue par substitution uniforme des atomes de A est aussi un élément de Λ.

Les éléments d'une logique sont appelés ses *théorèmes*. Nous écrirons $\vdash_\Lambda A$ pour signifier que la formule A est un théorème de la logique Λ; on a donc l'équivalence :

$$\vdash_\Lambda A \Longleftrightarrow A \in \Lambda.$$

Soit $\mathcal{C}$ un modèle ou une structure; la logique Λ est dite *adéquate* par rapport à $\mathcal{C}$ si tout théorème A de Λ est une formule valide dans $\mathcal{C}$, c'est-à-dire si l'on a :

$$\vdash_\Lambda A \Rightarrow \mathcal{C} \models A.$$

Une logique Λ est dite *complète* par rapport à $\mathcal{C}$ si toute formule A valide dans $\mathcal{C}$ est un théorème de la logique Λ, c'est-à-dire si l'on a :

$$\mathcal{C} \models A \Rightarrow \vdash_\Lambda A.$$

La logique Λ est dite *déterminée* par $\mathcal{C}$ si elle est à la fois adéquate et complète par rapport à $\mathcal{C}$, c'est-à-dire si l'on a :

$$\vdash_\Lambda A \Longleftrightarrow \mathcal{C} \models A.$$

2.2.6.3. *Logiques normales*

Une logique Λ est *normale* si elle contient le schéma de formule :

$$K \ : \ \Box(A \supset B) \supset (\Box A \supset \Box B),$$

et si elle est dotée de la *règle d'inférence modale de nécessité* (appelée en anglais *necessitation rule*), c'est-à-dire si l'on a :

$$\vdash_\Lambda A \Rightarrow \vdash_\Lambda \Box A.$$

Cette propriété énonce que si A est un théorème de la logique Λ, alors $\Box A$ est un théorème de cette logique Λ; ceci s'écrira :

$$\frac{\vdash_\Lambda A}{\vdash_\Lambda \Box A}.$$

Si $\{\Lambda_i \mid i \in I\}$ est un ensemble de logiques normales, alors $\cap\{\Lambda_i \mid \in I\}$ est une logique normale. En particulier la logique K définie par :

$$K = \cap\{\Lambda_i \mid \Lambda_i \text{ est une logique normale}\},$$

est la plus petite logique normale. Le théorème suivant, démontré dans [Goldblatt 87], montre l'importance de cette logique "minimale".

Théorème 2.2. Une formule A est un théorème de la logique K si et seulement si A est valide (c'est-à-dire valide dans toutes les structures).

2.2.6.4. *Systèmes axiomatiques*

Un *système axiomatique* pour une logique normale possède les trois composantes suivantes :

- Un système axiomatique de la logique propositionnelle.
- Le schéma d'axiome K : $\Box(A \supset B) \supset (\Box A \supset \Box B)$.
- La règle d'inférence modale de nécessité.

Il est d'usage courant de représenter la plus petite logique normale Λ contenant les schémas $\Sigma_1, \cdots, \Sigma_n$ par la notation :

$$\Lambda = K\Sigma_1 \cdots \Sigma_n.$$

Cette logique est définie comme suit :

$$\Lambda = \cap\{\Lambda_i \mid \Lambda_i \text{ est normale et } \Sigma_1 \cup \cdots \cup \Sigma_n \subset \Lambda_i\}.$$

Propriétés d'une logique normale :

1. Si Λ est une logique normale, alors :
   (a) $\vdash_\Lambda (A \supset B) \Longrightarrow \vdash_\Lambda \Box A \supset \Box B$
   (b) $\vdash_\Lambda (A \supset B) \Longrightarrow \vdash_\Lambda \Diamond A \supset \Diamond B$
   (c) $\vdash_\Lambda (A \equiv B) \Longrightarrow \vdash_\Lambda \Box A \equiv \Box B$
   (d) $\vdash_\Lambda (A \equiv B) \Longrightarrow \vdash_\Lambda \Diamond A \equiv \Diamond B$
   (e) $\vdash_\Lambda \Diamond A \equiv \neg\Box\neg A$
   (f) $\vdash_\Lambda (\Box A \wedge \Box B) \equiv \Box(A \wedge B)$
   (g) $\vdash_\Lambda (\Box A \vee \Box B) \supset \Box(A \vee B)$
   (h) $\vdash_\Lambda \Diamond(A \vee B) \equiv (\Diamond A \vee \Diamond B)$
   (i) $\vdash_\Lambda \Diamond(A \wedge B) \supset (\Diamond A \wedge \Diamond B)$
2. Une logique est normale si et seulement si :

$$\frac{\vdash_\Lambda (A_1 \wedge \cdots \wedge A_n) \supset A}{\vdash_\Lambda (\Box A_1 \wedge \cdots \wedge \Box A_n) \supset \Box A}.$$

(Cette dernière propriété est souvent utilisée comme définition d'une logique normale.)

2.2.7. *Quelques logiques classiques*

Les logiques KT, $S4$ et $S5$ sont d'usage courant en intelligence artificielle; les deux dernières sont définies comme suit :

$S4$: $KT4$
$S5$: $KT45$

Théorème 2.3. Une formule A est un théorème de la logique KT si et seulement si A est valide dans toute structure où R est réflexive.

Théorème 2.4. Une formule A est un théorème de la logique $S4$ si et seulement si A est valide dans toute structure où R est réflexive et transitive.

Théorème 2.5. Une formule A est un théorème de la logique $S5$ si et seulement si A est valide dans toute structure où R est réflexive, transitive et euclidienne (ce qui signifie que R est une relation d'équivalence).

Considérons la formalisation des systèmes de connaissance et de croyance; l'opérateur modal $\Box$ y prend respectivement la signification "est connu" et "est cru".

Par définition, une connaissance est une information vraie; le schéma T demande que "ce qui est connu soit vrai". Ce schéma sera inclus dans le système axiomatique d'une logique normale lorsque l'on désire que l'opérateur $\Box$ signifie "est connu".

Par contre le schéma T ne figurera pas dans un système d'axiomes formalisant la croyance car une croyance peut parfois être erronée. Lorsque l'opérateur modal $\Box$ signifie "est connu", alors le schéma 4 signifie : "Si je sais que A est vrai, alors je sais que je sais que A est vrai".

La faculté d'introspection positive décrite par le schéma 4 est nécessaire à la formalisation d'une intelligence introspective parfaite. Le schéma 5 formalise une introspection négative parfaite. Ce schéma signifie : "Si je ne sais pas que A est vrai, alors je sais que je ne sais pas que A est vrai". Cette propriété exprime une parfaite connaissance de nos limites de connaissance.

On obtient des systèmes axiomatiques pour les logiques $KT, S4$ et $S5$ en incorporant respectivement les schémas T, 4 et 5 comme schémas d'axiomes dans un système axiomatique d'une logique normale.

Le choix d'un système modal dépendra du type de connaissance ou de croyance à modéliser.

Si l'on souhaite caractériser la connaissance d'un agent intelligent ayant une parfaite capacité d'introspection logique sur ce qu'il "connaît" et ne "connaît pas", on choisira le système modal $S5$.

Si l'on désire modéliser les croyances d'un agent idéalement rationnel (c'est-à-dire un agent dont certaines croyances peuvent se révéler erronées mais qui cependant possède une parfaite faculté d'introspection logique sur ce qu'il croit et ne croit pas), on choisira le système $K45$, appelé aussi *faible* $S5$.

Chacun de ces différents systèmes modaux induit une relation syntaxique d'inférabilité (de dérivabilité) qui lui est propre; elle est notée $\vdash_S$ où S est le nom du système modal considéré.

2.2.8. *Système $S5$: nécessité et connaissance introspective*

Le système $S5$ peut être défini au moyen d'un des systèmes axiomatiques équivalents suivants :

$$S5 = KTB4 = KDB4 = KDB5 = KT5.$$

Ces axiomes reçoivent respectivement les interprétations suivantes dans les logiques de nécessité et de connaissance :

T :	$\Box A \supset A$	:	Si A est nécessairement vrai, alors A est vrai.
		:	Si je sais que A est vrai, alors A est vrai.
5 :	$\Diamond A \supset \Box\Diamond A$	:	Si A est possiblement vrai, alors il est nécessaire que A soit possiblement vrai.
		:	Si je ne sais pas si A est vrai, alors je sais que je ne sais pas si A est vrai.
B :	$A \supset \Box\Diamond A$	:	Si A est vrai, alors il est nécessaire qu'il soit possible que A soit vrai.
		:	Si A est vrai, alors je sais qu'il est faux de savoir que l'inverse de A est vrai.
4 :	$\Box A \supset \Box\Box A$	:	Si A est nécessairement vrai, alors il est nécessaire que A soit nécessairement vrai.
		:	Si je sais A, alors je sais que je sais A.
D :	$\Box A \supset \Diamond A$	:	S'il est nécessaire que A soit vrai, alors il est possible que A soit vrai.
		:	Si je sais que A est vrai, alors je ne sais pas que l'inverse de A est vrai.

Les diverses propriétés de la relation d'accessibilité ne sont pas nécessairement indépendantes; par exemple, la propriété d'euclidianité découle de la transitivité et de la symétrie; ceci s'écrit :

$$\begin{array}{ll} 5: & \forall s \forall t \forall u (sRt \wedge sRu \supset tRu) \\ B: & \forall s \forall t (sRt \supset tRs) \\ \hline 4: & \forall s \forall t \forall u (tRs \wedge sRu \supset tRu) \end{array}$$

La dérivation de la propriété 4 à partir des propriétés 5 et B est triviale; une dérivation équivalente est possible dans le formalisme de la logique à partir des schémas d'axiomes, soit :

$$\begin{array}{ll} 5: & \neg\Box A \supset \Box\neg\Box A \\ B: & A \supset \Box\neg\Box\neg A \\ \hline 4: & \Box A \supset \Box\Box A \end{array}$$

La dérivation à partir du formalisme logique est toujours nettement plus ardue qu'à partir des propriétés des relations. La dérivation du schéma 4 au moyen des schémas 5 et B en constitue une illustration.

Un système axiomatique possible pour $S5$ est le suivant :

Système axiomatique :

T : $\Box A \supset A$
5 : $\Diamond A \supset \Box\Diamond A$
K : $\Box(A \supset B) \supset (\Box A \supset \Box B)$
Df : $\Diamond A \equiv \neg\Box\neg A$
PL : A est une tautologie
RN : $\frac{\vdash A}{\vdash \Box A}$

$$MP : \frac{\vdash A \supset B, \vdash A}{\vdash B}$$

Le schéma 4 est un théorème dérivable de ce système axiomatique. Pour le démontrer, nous établissons trois théorèmes intermédiaires dénommés respectivement $T_\Diamond$, $B_\Diamond$ et *RM*.

Théorème $T_\Diamond$. Le schéma :

$$T_\Diamond : A \supset \Diamond A$$

(ce qui est vrai est possiblement vrai) est un théorème de $S5$.
Démonstration.

1.	$\Box\neg A \supset \neg A$	réécriture du schéma T
2.	$A \supset \neg\Box\neg A$	1 et règle propositionnelle
3.	$\neg\Box\neg A \equiv \Diamond A$	définition de $\Diamond A$
4.	$A \supset \Diamond A$	2, 3, et règle propositionnelle

Théorème $B_\Diamond$. Le schéma :

$$B_\Diamond : A \supset \Box\Diamond A$$

(ce qui est vrai est nécessairement possiblement vrai) est un théorème de $S5$.
Démonstration.

1.	$\Diamond A \supset \Box\Diamond A$	5
2.	$A \supset \Diamond A$	$T_\Diamond$
3.	$A \supset \Box\Diamond A$	1, 2, et règle propositionnelle

Théorème *RM*. La règle :

$$RM : \frac{\vdash (A \supset B)}{\vdash (\Box A \supset \Box B)}$$

est une règle d'inférence de $S5$.
Démonstration.

1.	$A \supset B$	hypothèse que 1 est un théorème
2.	$\Box(A \supset B)$	1, *RN*
3.	$\Box(A \supset B) \supset (\Box A \supset \Box B)$	*K*
4.	$\Box A \supset \Box B$	2, 3, et règle propositionnelle

Théorème *4*. Le schéma :

$$4 : \Box A \supset \Box\Box A$$

est un théorème de $S5$.

Démonstration.

1.	$\Diamond A \supset \Box\Diamond A$	5
2.	$\Diamond\neg A \supset \Box\Diamond\neg A$	1 et règle propositionnelle
3.	$\neg\Box\Diamond\neg A \supset \neg\Diamond\neg A$	2 et règle propositionnelle
4.	$\Diamond\neg\Diamond\neg A \supset \neg\Diamond\neg A$	3, *Df*
5.	$\Diamond\Box A \supset \Box A$	*Df*
6.	$\Box\Diamond\Box A \supset \Box\Box A$	5, *RM*
7.	$A \supset \Box\Diamond A$	$B_\Diamond$
8.	$\Box A \supset \Box\Diamond\Box A$	7, et règle propositionnelle
9.	$\Box A \supset \Box\Box A$	6, 8, et règle propositionnelle

On montre aisément que les axiomes de $S5$ sont *valides dans la structure*, que les règles d'inférence préservent la validité et que donc l'axiomatisation est *adéquate*. Elle est également complète et décidable.

2.2.9. *Les logiques obtenues à partir des schémas D, T, B, 4, 5*

Il existe quinze systèmes axiomatiques distincts obtenus à partir de la logique K en ajoutant au système axiomatique un sous-ensemble des schémas d'axiome D, T, B, 4, 5. Ces systèmes sont représentés à la figure 2.2.

Une *modalité* est une séquence d'opérateurs $\neg$, $\Box$, et $\Diamond$, y compris la séquence vide notée $\odot$. Une *modalité positive* ne contient pas l'opérateur $\neg$. Les modalités positives pour les systèmes $KT5$, $KT4$, $K5$, $KD5$, $K45$, $KB4$ et $KD45$ sont respectivement représentées aux figures 2.3–2.9

Le système $S5$ contient six modalités, à savoir :

Modalités positives :	$\odot$	$\Box$	$\Diamond$
Modalités négatives :	$\neg$	$\neg\Box$	$\neg\Diamond$

Ceci peut se démontrer en observant que $S5$ contient les schémas suivants, interprétables comme lois de réduction :

$$\begin{aligned} 1 &: \Box A \equiv \Box\Box A \\ 2 &: \Diamond A \equiv \Diamond\Diamond A \\ 3 &: \Box A \equiv \Diamond\Box A \\ 4 &: \Diamond A \equiv \Box\Diamond A \end{aligned}$$

D'autres systèmes possèdent également un nombre fini de modalités.
Voici quelques théorèmes généraux concernant les systèmes normaux.

Théorème 2.6. Les 15 systèmes normaux de la figure 2.2 sont tous distincts.

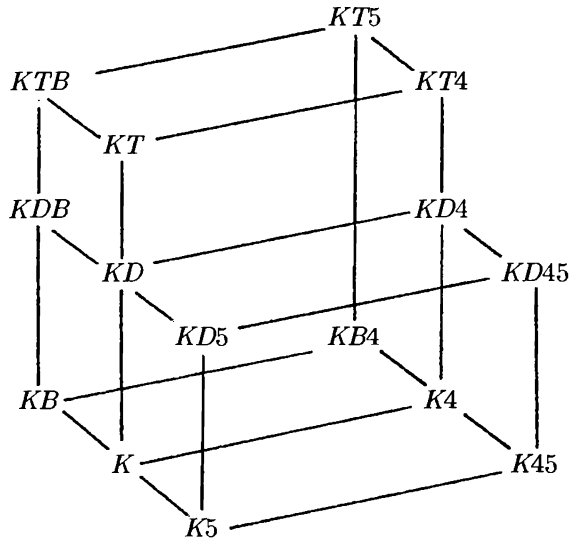


Figure 2.2 *Systèmes axiomatiques obtenus à partir de la logique K*

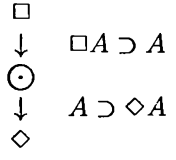


Figure 2.3 *Modalités positives de S5*

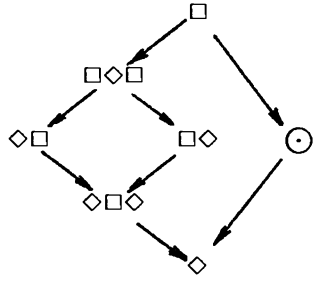


Figure 2.4 *Modalités positives de KT4*

Théorème 2.7. Les logiques normales contenant un sous-ensemble des schémas 1–10 sont adéquates et complètes pour les structures satisfaisant aux propriétés correspondantes.

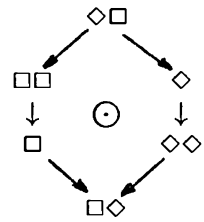


Figure 2.5 *Modalités positives de K5*

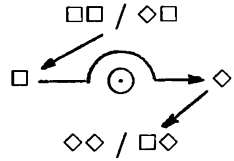


Figure 2.6 *Modalités positives de KD5*

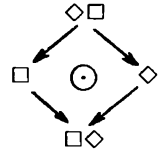


Figure 2.7 *Modalités positives de K45*

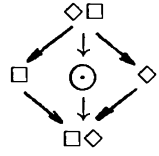


Figure 2.8 *Modalités positives de KB4*

Figure 2.9 *Modalités positives de KD45*

Théorème 2.8. Les systèmes *KT*5, *KT*4, *K*5, *KD*5, *K*45, *KB*4 et *KD*45 ont exactement le nombre de modalités positives représentées aux figures 2.3–2.9.

Théorème 2.9. Chacun des systèmes *K*, *KD*, *KT*, *KB*, *K*4, *KDB*, *KD*4 et *KTB* a un nombre infini de modalités.

Théorème 2.10. Chacun des 15 systèmes de la figure 2.2 est décidable.

2.2.10. *Langages multimodaux*

Les logiques utilisant plus d'un opérateur modal sont appelées *multimodales.* Les analyses des propriétés du "temps" ont conduit à la définition de logiques temporelles qui sont naturellement multimodales puisque le temps possède deux composantes : le futur et le passé.

Les langages multimodaux utilisent une collection de symboles $\{[\, i\,] \mid i \in I\}$ dont chacun correspond à un opérateur universel. L'opérateur dual, noté $<i>$, est défini comme $\neg[\, i\,]\neg$; les logiques utilisant plus d'un opérateur modal sont appelées *multimodales.*

SYNTAXE

- Toutes les règles syntaxiques de la logique des propositions sont aussi des règles syntaxiques de la logique multimodale.
- Si A est une formule, alors $[\, i\,]A$ et $<i>A$ sont des formules.

Une *structure* $\mathcal{F}$ pour un langage multimodal est définie comme suit :

$$\mathcal{F} = <W, \{R_i \mid i \in I\}>,$$

où W est un ensemble non vide et $\{R_i\}$ est une collection de relations binaires $R_i \subset W \times W$. Un *modèle* $\mathcal{M} = <\mathcal{F}, V>$ sur une structure $\mathcal{F}$ est défini comme précédemment au moyen d'une application V de l'ensemble P des atomes du langage, dans 2^W. La sémantique d'un langage multimodal est définie comme au paragraphe 2.2.3, sauf que la dernière règle est remplacée par l'ensemble suivant de règles (avec $i \in I$).

SÉMANTIQUE

$$\mathcal{M} \models_s [\, i\,]A \text{ ssi } sR_it \text{ implique } \mathcal{M} \models_t A \text{ pour tout } t \in W.$$

Les définitions de *formule* (A) *valide dans un modèle* ($\mathcal{M} \models A$), *valide dans une structure* ($\mathcal{F} \models A$) et *valide* ($\models A$) demeurent inchangées. De même une logique Λ est encore définie comme un sous-ensemble d'un langage multimodal qui contient toutes les tautologies et qui est fermé pour les règles de *modus ponens* et de *substitution uniforme.*

Une logique Λ est *normale* si elle contient les schémas :

$$K_i \ : [\, i\,](A \supset B) \supset ([\, i\,]A \supset [\, i\,]B),$$

et si elle est dotée des règles de nécessité :

$$\vdash_\Lambda A \Rightarrow \vdash_\Lambda [\, i\,]A,$$

pour tout $i \in I$. La plus petite logique normale est notée K_I.

2.2.11. *Exemples de langages multimodaux*

Un langage combinant les modalités temporelles "dans le *futur*" et "dans le *passé* " est *multimodal.* Ses modalités atomiques sont les suivantes :

$[F]$: toujours dans le futur; $<F>$: parfois dans le futur;
$[P]$: toujours dans le passé; $<P>$: parfois dans le passé.

Les langages de ce type sont étudiés dans la section 3.2, entièrement consacrée à la logique temporelle.

Les langages qui décrivent les connaissances et les croyances de diverses personnes, qualifiées d'"agents", sont des langages multimodaux; la modalité "l'agent i sait que" (ou "croit que") est représentée par la modalité atomique $[K_i]$.

Par exemple, si la proposition p signifie "il pleut", que cette proposition est *vraie*, et que ni l'agent 1, ni l'agent 2, ne savent qu'il pleut, et si de plus l'agent 2 ne sait pas si l'agent 1 sait qu'il pleut ou sait qu'il ne pleut pas, alors la formule suivante est *vraie* :

$$p \wedge \neg[K_1]p \wedge \neg[K_2]p \wedge \neg[K_2]([K_1]p \vee [K_1]\neg p).$$

Les langages décrivant les connaissances et les croyances d'agents rentrent dans le cadre de la logique épistémique étudiée dans la section 3.3 de ce volume.

2.2.12. *La logique dynamique*

La logique dynamique est une extension d'une logique multimodale. Le langage de la logique dynamique découle de l'idée d'associer à chaque commande α d'un langage de programmation, un opérateur modal $[\alpha]$ tel que la formule $[\alpha]A$ signifie : "Après chaque fin d'exécution de α, la formule A est vraie". La formule duale $<\alpha>A$ signifie : "Il y a au moins une exécution de α qui se termine avec la formule A vraie". Ce type d'opérateur modal permet de construire une théorie qui énonce formellement des propriétés de programmes au moyen d'opérateurs modaux et de programmes élémentaires, encore appelés "programmes atomiques". Par rapport aux langages logiques étudiés précédemment, le langage de la logique dynamique utilise un ensemble de symboles de base supplémentaires α qui représentent les *programmes atomiques* [Goldblatt 87].

Soit Π l'ensemble des programmes atomiques, et soit P l'ensemble des propositions du langage.

SYNTAXE

- Toutes les règles syntaxiques de la logique multimodale sont aussi des règles syntaxiques de la logique dynamique; les opérateurs modaux y sont $[\alpha]$ et $<\alpha>$, où α est un programme.
- Syntaxe des programmes
  - **Base** : tout programme atomique est un programme.
  - **Induction** : si α, α_1 et α_2 sont des programmes et si A est une formule, alors : $\alpha_1;\alpha_2$, $\alpha_1 \cup \alpha_2$, α^* et $A?$ sont des programmes.
  - **Fermeture** : un programme s'obtient uniquement à partir des règles de base et d'induction.

On interprète les opérations sur les programmes de la façon suivante :

Programme		Signification
$\alpha_1;\alpha_2$	:	Exécuter α_1 et ensuite α_2.
$\alpha_1 \cup \alpha_2$	:	Exécuter soit α_1, soit α_2 (non-déterminisme).
α^*	:	Exécuter α un nombre fini de fois.
$A?$	:	Tester A : si A est vrai continuer, sinon interrompre.

Ces opérations élémentaires permettent de définir les instructions classiques des langages de programmation :

if A **then** α **else** β $=_{def}$ $(A?;\alpha) \cup (\neg A?;\beta)$
while A **do** α $=_{def}$ $(A?;\alpha)^*;\neg A?$
repeat α **until** A $=_{def}$ $\alpha\ ;\ (\neg A?;\alpha)^*$

Un modèle $\mathcal{M}$ pour le langage de la logique dynamique est un triplet :

$$\mathcal{M} = <W, \{R_\alpha \mid \alpha \in \textit{programmes}(P,\Pi)\}, V>,$$

où R_α est une relation binaire sur W pour chaque programme α, et où V est une application de P dans 2^W.

SÉMANTIQUE

$\mathcal{M} \models_s [\alpha]A$ ssi $sR_\alpha t$ implique $\mathcal{M} \models_t A$.

La relation binaire R_α doit évidemment refléter les significations des programmes α. Un modèle $\mathcal{M}$ est dit *standard* si la relation binaire satisfait aux conditions suivantes :

$R_{\alpha;\beta} = \{(s,t) \mid \exists u(sR_\alpha u \wedge uR_\beta t)\}$
$R_{\alpha\cup\beta} = R_\alpha \cup R_\beta$
$R_{\alpha^*} = (R_\alpha)^*$ (la fermeture réflexive et transitive de R_α)
$R_{A?} = \{(s,s) \mid \mathcal{M} \models_s A\}$

Considérons la plus petite logique normale contenant les schémas d'axiome suivants :

AXIOMES
$[\alpha;\beta]A \equiv [\alpha][\beta]A$
$[\alpha \cup \beta]A \equiv [\alpha]A \wedge [\beta]A$
$[A?]B \equiv (A \supset B)$
$[\alpha^*]A \supset (A \wedge [\alpha][\alpha^*]A)$
$[\alpha^*](A \supset [\alpha]A) \supset (A \supset [\alpha^*]A)$

On peut montrer que cette logique est déterminée par la classe des modèles standards.

2.2.13. *Logique temporelle et logiques de nécessité*

Les logiques de nécessité sont issues de l'antiquité grecque : on considère essentiellement les écoles stoïque et mégarique.

Ecole stoïque :

Ce qui est *possible* est ce qui est vrai (réalisé) au temps présent ou dans le futur :

$$D_{\Diamond 1} : \Diamond A \equiv A \vee \langle F \rangle A.$$

Ce qui est *nécessaire* est ce qui est vrai (réalisé) au temps présent et à chaque moment dans le futur :

$$D_{\Box 1} : \Box A \equiv A \wedge [F]A.$$

Ecole mégarique :

Ce qui est *possible* est ce qui est vrai (réalisé) à un temps quelconque :

$$D_{\Diamond 2} : \Diamond A \equiv A \vee \langle F \rangle A \vee \langle P \rangle A.$$

Ce qui est *nécessaire* est ce qui est vrai (réalisé) à tous les temps :

$$D_{\Box 2} : \Box A \equiv A \wedge [F]A \wedge [P]A.$$

On peut démontrer que le système axiomatique constitué de l'axiome $D_{\Box 1}$ et de ceux de la logique K_t est équivalent au système M de Feys (ou logique KD). Ce système est défini axiomatiquement comme suit :

D : $\Box A \supset A$
K : $\Box(A \supset B) \supset (\Box A \supset \Box B)$
RN : $\vdash A \Rightarrow \vdash \Box A$

2.2.14. *Logique déontique*

La logique déontique est la branche de la logique qui concerne le raisonnement à propos de normes.

Les opérateurs de la logique déontique modélisent les modalités de la *permission* ($[Pe]$), de l'*obligation* ($\bigcirc$) et de l'*interdiction* ($[Fo]$) (forbidden).

Voici quelques systèmes axiomatiques classiques de logique déontique [Meyer and Wieringa 93].

2.2.14.1. *Le système OS de Von Wright*

Ce système date de 1951; OS signifie "old system".

$OS0$: tautologies propositionnelles
$OS1$: $\bigcirc A \equiv \neg[Pe]\neg A$
$OS2$: $[Pe]A \vee [Pe]\neg A$
$OS3$: $[Pe](A \vee B) \equiv [Pe]A \vee [Pe]B$
$OS4$: $\frac{\vdash A \equiv B}{\vdash [Pe]A \equiv [Pe]B}$

On observe que le système OS est fort proche d'une logique normale. On peut démontrer que le système OS est équivalent au système KD pourvu que l'on introduise l'axiome supplémentaire :

OT : $\bigcirc(A \vee \neg A)$.

Le système de Von Wright peut alors être représenté sous la forme d'un système de la logique normale.

$KD0$: tautologies propositionnelles
$KD1$: $\bigcirc(A \supset B) \supset (\bigcirc A \supset \bigcirc B)$
$KD2$: $\bigcirc A \supset [Pe]A$
$KD3$: $[Pe]A \equiv \neg \bigcirc \neg A$
$KD4$: $[Fo]A \equiv \neg[Pe]\neg A$
$KD5$: *modus ponens* : $\frac{\vdash A, \vdash A \supset B}{\vdash B}$
$KD6$: *nécessité* : $\frac{\vdash A}{\vdash \bigcirc B}$

La logique résultante est appelée *système standard de la logique déontique*, ou système KD.

La loi de nécessité ($KD6$) implique l'axiome :

OT : $\bigcirc(A \vee \neg A)$,

qui avait été rejeté par Von Wright.

Voici quelques théorèmes caractéristiques de la logique KD :

$\bigcirc(A \wedge B) \equiv \bigcirc A \wedge \bigcirc B$
$\neg(\bigcirc A \wedge \bigcirc \neg A)$
$[Pe](p \vee q) \equiv [Pe]A \vee [Pe]B$
$\bigcirc A \supset \bigcirc(A \vee B)$
$[Fo]A \supset [Fo](A \wedge B)$
$\bigcirc A \supset \bigcirc(A \supset B)$
$\bigcirc \neg A \supset \bigcirc(B \supset A)$
$\neg A \supset (A \supset \bigcirc B)$
$\neg \bigcirc (A \wedge \neg A)$

2.2.14.2. *Le système NS (Von Wright 1964) : approche dyadique*

Le système NS (nouveau système) proposé par Von Wright tient compte des *obligations conditionnelles*. Pour ce faire, la syntaxe du système NS contient un opérateur supplémentaire :

$\bigcirc(A/B)$: A est obligatoire sous la condition B.

Un système axiomatique pour NS est le suivant.

$NS0$: tautologies propositionnelles
$NS1$: $\bigcirc(A \wedge B/C) \equiv \bigcirc(A/C) \wedge \bigcirc(B/C)$
$NS2$: $\bigcirc(A/B \vee C) \equiv \bigcirc(A/B) \wedge \bigcirc(B/C)$
$NS3$: $\neg(\bigcirc(A/B) \wedge \bigcirc(\neg A/B))$
$NS4$: $[Pe](A/B) \equiv \neg \bigcirc (\neg A/B)$
$NS5$: $\bigcirc A \equiv \bigcirc(A/(B \supset B))$
$NS6$: *modus ponens*
$NS7$: *substitution uniforme*

On peut démontrer que le système NS est une extension du système OS.

2.2.14.3. *Réduction à la logique modale aléthique (Anderson 1958)*

Anderson a proposé de réduire la logique déontique à la logique aléthique, c'est-à-dire à une logique sans contenu déontique, sauf pour un atome particulier V dont la signification est quelque chose comme : une *responsabilité*, un *risque à courir*, une *amende*, une *sanction*, une *punition*.

Le système qui en résulte étend le système modal *KT*.

($A0$) : Axiomes de la logique propositionnelle
($A1$) : $\Box A \supset A$
($A2$) : $\Box(A \supset B) \supset (\Box A \supset \Box B)$
($A3$) : $\Diamond A \equiv \neg\Box\neg A$
($A4$) : $\neg\Box \mathtt{V}$
($A5$) : $\bigcirc A \equiv \Box(\neg A \supset \mathtt{V})$
($A6$) : $[Pe]A \equiv \Diamond(A \wedge \neg \mathtt{V})$
($A7$) : $[Fo]A \equiv \Box(A \supset \mathtt{V})$
($A8$) : *modus ponens*
($A8$) : *substitution uniforme*
($A8$) : *nécessité*

Voici les interprétations de certains de ces axiomes :

($A4$) : $\neg\Box \mathtt{V}$: La sanction n'est pas nécessaire; il existe un monde où on n'est pas puni; il reste toujours de l'espoir.

($A5$) : $\bigcirc A \equiv \Box(\neg A \supset \mathtt{V})$: L'obligation de faire A est équivalente à l'assertion que "ne pas faire A" implique nécessairement une sanction.

($A6$) : $[Pe]A \equiv \Diamond(A \wedge \neg \mathtt{V})$: A est permis si et seulement s'il est nécessaire qu'il soit possible d'effectuer A sans encourir de sanction.

($A7$) : $[Fo]A \equiv \Box(A \supset \mathtt{V})$: A est défendu si et seulement si le fait de faire A implique nécessairement une sanction.

Voici quelques théorèmes controversés du système :

$\bigcirc A \supset \bigcirc \bigcirc A$: l'obligation de A entraîne l'obligation de l'obligation de A.
$\Box(\Box A \supset \bigcirc A)$: la nécessité de A entraîne nécessairement l'obligation de A.
$\Box(\bigcirc A \supset \Box \bigcirc A)$: l'obligation de A entraîne nécessairement que l'obligation de A soit nécessaire.
$\Box(A \supset B) \supset \Box(\bigcirc A \supset \bigcirc B)$: paradoxe du bon Samaritain.

On illustre le paradoxe du bon Samaritain comme suit :

A : Le bon Samaritain aide Jean qui a été blessé.
B : Jean a été blessé.
$\Box(A \supset B) \supset \Box(\bigcirc A \supset \bigcirc B)$:
S'il est nécessairement vrai que : si le bon Samaritain aide Jean qui a été blessé implique que Jean a été blessé,

alors il est nécessaire que : il faudrait que le bon Samaritain aide Jean qui a été blessé implique qu'il faudrait que Jean soit blessé.

Donc si le bon Samaritain rencontre sur son chemin une personne qui n'est pas blessée, il doit la blesser, pour ensuite pouvoir la secourir; le sens supposé de la parabole n'est pas rendu par l'analyse logique.

2.2.14.4. *Approche dynamique*

La logique déontique dynamique utilise des opérateurs $[\alpha]$ pour chaque programme α; dans cette interprétation l'expression $[\alpha]A$ signifie : l'exécution du programme α conduit nécessairement à un état (un monde possible) où la formule A est vérifiée. Nous avons les axiomes suivants :

$[Fo]\alpha \equiv [\alpha]\mathtt{V}$: l'action α est interdite si et seulement si l'exécution de α conduit à un état où existe une sanction.
$[Pe]\alpha \equiv \neg[Fo]\alpha \equiv <\alpha>\neg\mathtt{V}$: l'action α est permise si et seulement si elle n'est pas interdite (si il y a au moins une façon d'effectuer α qui conduise à un état où il n'y a pas de sanction).
$\bigcirc\alpha \equiv [Fo](-\alpha) \equiv [-\alpha]\mathtt{V}$: l'action α est obligatoire si et seulement si ne pas faire α (ce qui est représenté par $-\alpha$) est interdit.

2.3. La sémantique algébrique

2.3.1. *Introduction*

Nous proposons dans cette section un aperçu d'une deuxième approche sémantique pour la logique modale, l'approche dite *algébrique*.

La logique propositionnelle est *vérifonctionnelle* : la valeur de vérité attribuée à une formule dépend uniquement de la structure de cette formule et des valeurs de vérité attribuées aux propositions qu'elle contient.

Les connecteurs du calcul des propositions sont donc *vérifonctionnels*; ainsi, la valeur de vérité associée à la formule $(X \wedge Y)$ est connue dès que les valeurs de vérité associées aux formules X et Y le sont.

Un connecteur vérifonctionnel est caractérisé par sa table de vérité. On peut se demander s'il est possible d'étendre le traitement vérifonctionnel de la logique propositionnelle aux opérateurs de la logique modale.

Il existe quatre fonctions de vérité unaires, indiquées dans la figure 2.10 :

Valeur	identité	négation	tautologie	contradiction
0	0	1	1	0
1	1	0	1	0

Figure 2.10 *Fonctions de vérité unaires*

Donc si les modalités du *nécessaire* et du *possible* étaient des fonctions vérifonctionnelles unaires, ces fonctions devraient être une des quatre indiquées dans la figure 2.10, ce qui manifestement ne peut être le cas.

Si l'on veut à tout prix proposer une analyse vérifonctionnelle du *nécessaire* et du *possible*, il faut donc abandonner le principe d'une logique à deux valeurs, et se situer dans le cadre de logiques à plusieurs valeurs, ou *logiques multivaluées*.

2.3.2. *La logique à trois valeurs de Łukasiewicz*

Quels changements doit-on apporter à la logique bivalente si l'on veut introduire les modalités du "possible" et du "nécessaire"? On doit tout d'abord considérer plusieurs classes de propositions.

Appelons N la classe des propositions "nécessaires", P celle des propositions "possibles", I celle des propositions "impossibles" ou "absurdes", C celle des propositions "contingentes" ou "éventuellement fausses".

Aucune proposition n'appartient en même temps à N et C, ni à I et P; c'est un principe de contradiction. La classe N est incluse dans la classe P, et la classe I dans la classe C. Ceci s'énonce sous la forme d'un principe de l'implication du possible par le nécessaire et du contingent par l'absurde :

> *Toute proposition nécessaire est possible, toute proposition absurde n'est pas nécessaire.*

Il y a des propositions qui sont possibles et contingentes. Ce sont les propositions "problématiques"; appelons U l'ensemble de celles-ci. On voit maintenant apparaître un principe du quart exclu :

> *Toute proposition appartient soit à la classe N, soit à la classe U, soit à la classe I.*

Cette théorie de la modalité peut se traduire sous forme algébrique. (Voir à ce sujet le traité de Moisil [Moisil 72].)

A chacune des classes N, U, I correspond une interprétation : le "nécessaire", le "problématique", l'"absurde". Considérons trois symboles 2,1,0 qui sont des valeurs logiques désignant ces interprétations; à chaque proposition on peut faire correspondre sa valeur logique.

Cette logique à trois valeurs (ou *logique trivalente*) est due à Łukasiewicz; les articles originaux, publiés en 1920 et 1930, sont écrits en polonais et en allemand; ils sont rassemblés dans [Łukasiewicz 70].

Dans la logique de Łukasiewicz chaque proposition possède une des valeurs 0, 1 ou 2 (interprétée comme la valeur sémantique de la proposition). Les valeurs sémantiques sont celles indiquées dans les figures 2.11 et 2.12.

Ces figures définissent la sémantique pour une logique modale du possible et du nécessaire au moyen d'une logique trivalente. (Voir [Rescher 69], p. 22–28; [Turner 84].)

p	$\neg p$
2	0
1	1
0	2

$p \setminus q$	2	1	0
2	2	1	0
1	1	1	0
0	0	0	0

$p \wedge q$			$p \vee q$			$p \supset q$			$p \equiv q$		
2	1	0	2	1	0	2	1	0	2	1	0
2	1	0	2	2	2	2	1	0	2	1	0
1	1	0	2	1	1	2	2	1	1	2	1
0	0	0	2	1	0	2	2	2	0	1	2

Figure 2.11 *Tables ternaires (Connecteurs de Lukasiewicz)*

p	$\Diamond p$	$\Box p$
2	2	2
1	2	0
0	0	0

Figure 2.12 *Table ternaire des opérateurs modaux*

2.3.3. *Les logiques multivaluées*

Représentons par $V(p)$ la valeur de vérité de la formule p. La logique trivalente de Lukasiewicz peut être généralisée à un nombre indéterminé de valeurs comprises entre 0 et m (qui jouent respectivement les rôles des constantes sémantiques **F** et **V**). Ces valeurs de vérité sont généralement les entiers : $0, \cdots, m$; la sémantique des différents connecteurs peut être définie comme suit :

$$\begin{aligned} V(\neg p) &= m - V(p); \\ V(p \vee q) &= max\,(V(p), V(q)); \\ V(p \wedge q) &= min\,(V(p), V(q)); \\ V(p \supset q) &= m \text{ si } V(p) \leq V(q); \\ &= 0 \text{ si } V(p) > V(q)); \\ V(p \equiv q) &= min\,(V(p \supset q), V(q \supset p)). \end{aligned}$$

Une définition différente qui peut être utilisée pour l'implication est :

$$V(p \supset q) = max\,(V(\neg p), V(q)).$$

Il est classique de prendre la négation ($\neg$) et l'implication ($\supset$) comme primitives et de définir la conjonction et la disjonction comme suit :

$$p \vee q =_{def} (p \supset q) \supset q;$$
$$p \wedge q =_{def} \neg(\neg p \vee q).$$

L'intérêt des logiques multivaluées réside dans la possibilité de relier leurs valeurs sémantiques à diverses interprétations des modalités. Relevons quelques exemples d'interprétations avec leurs valeurs sémantiques associées :

Dans la table de la figure 2.13, les valeurs de vérité ont été partitionnées en :

1. Valeurs distinguées comme *vraies*, et notées "+".
2. Valeurs distinguées comme *fausses*, et notées "–".

		Interprétations	
		Ontologique	*Probabiliste*
(+)	4 :	nécessairement vrai	certainement vrai
(+)	3 :	présentement vrai	probablement vrai
(+/−)	2 :	indéterminé	indiscernable
(−)	1 :	présentement faux	probablement faux
(−)	0 :	nécessairement faux	certainement faux

Figure 2.13 *Interprétations pour une logique à cinq valeurs*

Ceci permet, par exemple, de définir une *tautologie* dans le contexte d'une logique multivaluée comme une formule qui prend le vrai comme *valeur distinguée*, quelles que soient les valeurs des atomes qui la constituent.

Une autre interprétation est celle où les valeurs de vérité reflètent le statut *épistémique* des propositions, par exemple :

1. Proposition que l'on *sait* être vraie, ou que l'on peut *prouver* être vraie.
2. Proposition dont *ne connaît pas* la valeur de vérité, ou que l'on ne peut *pas prouver* être vraie ou fausse.
3. Proposition que l'on *sait* être fausse, ou que l'on peut *prouver* être fausse.

On peut encore assigner des valeurs de vérité à une proposition p en référence à un ensemble d'axiomes :

1. La proposition p est dérivable à partir des axiomes.
2. La proposition *non-p* est dérivable à partir des axiomes.
3. Ni la proposition p, ni la proposition *non-p* ne sont dérivables à partir des axiomes.

On observe que dans ce dernier type d'interprétation, la valeur de vérité est *fonction des hypothèses* (des *axiomes*).

2.3.4. *Sémantiques pour des logiques à quatre valeurs*

On peut diviser l'ensemble des énoncés vrais en "contingentement vrais" (ceux qui sont vrais mais qui auraient pu être faux si le monde avait évolué autrement; les événements historiques sont des exemples d'événements contingents) et "nécessairement vrais" (ceux que l'on ne peut nier sans remettre en question la signification normale des mots du langage ou des vérités mathématiques ou logiques). De la même façon, les énoncés faux sont divisés en "contingentement faux" et "nécessairement faux". On suppose qu'une "vérité nécessaire" (dans le monde actuel) est associée à un événement qui, non seulement se produit dans le monde réel, mais également dans tous les mondes accessibles à partir de ce monde. De même, une "vérité contingente" est associée à un événement qui est vérifié dans le monde actuel mais ne l'est pas

nécessairement dans *tous* les mondes accessibles. Un énoncé "nécessairement faux" ne sera vérifié dans aucun des mondes accessibles tandis qu'un énoncé "contingentement faux" sera faux dans le monde actuel mais ne le sera pas nécessairement dans *tous* les mondes accessibles.

Ceci conduit à une première interprétation de la logique à quatre valeurs; une seconde interprétation est en termes probabilistes :

	Première interprétation	*Seconde interprétation*
3	nécessairement vrai	certainement vrai
2	contingentement vrai	probablement vrai
1	contingentement faux	probablement faux
0	nécessairement faux	certainement faux

On peut encore songer à construire une logique à quatre valeurs dont les significations sont les suivantes (logique de Prior, 1967) :

3	vrai et purement formel	$(2 + 2 = 4)$
2	vrai mais relativement à des faits	(la marquise sort à 5h)
1	faux mais relativement à des faits	(les impôts diminuent)
0	faux et purement formel	$(2 + 2 = 5)$

Selon cette nomenclature, on classera les propositions en *pures*, c'est-à-dire strictement formelles (abstraites ou mathématiques), et *impures*, c'est-à-dire se référant à des faits relatifs à des événements ou objets naturels. Le caractère *impur* contamine toute autre proposition associée conjonctivement avec une proposition pure. Par exemple, "les impôts diminuent et $(2 + 2 = 5)$" prend la valeur "1" même si l'énoncé "$2 + 2 = 5$" est faux.

On peut enfin proposer une quatrième interprétation :

3 vrai et a priori
2 vrai et a posteriori
1 faux et a posteriori
0 faux et a priori

Les philosophes considèrent aujourd'hui comme coextensifs (mais évidemment pas comme synonymes) les prédicats : "analytique" et "a priori", "synthétique" et "a posteriori". La thèse quantienne de l'astuce de jugements synthétiques a priori a été abandonnée. Kripke a reconsidéré les rapports entre "a priori" et "nécessaire", "a posteriori" et "contingent", et défendu la thèse de l'existence d'énoncés nécessaires a posteriori. Il donne comme exemple "la chaleur est un mouvement des molécules".

2.3.5. *Logiques multivaluées codées binaires*

Nous pouvons décider de coder la sémantique d'une logique tétravalente, dont la sémantique modélise ces notions de *pur* et d'*impur*, de la façon suivante :

1 : vrai; 0 : faux;
1 : purement formel; 0 : non purement formel.

Ce codage fournit les correspondances :

3 : vrai et purement formel : (1,1)
2 : vrai et non purement formel : (1,0)
1 : faux et non purement formel : (0,0)
0 : faux et purement formel : (0,1)

La conjonction de deux formules logiques codées $p = (\alpha, \beta)$ et $q = (\gamma, \delta)$, est alors définie composante par composante :

$$(\alpha, \beta) \wedge (\gamma, \delta) = (\alpha \wedge \gamma, \beta \wedge \delta).$$

$p \wedge q$

$p \setminus q$	3 : (1,1)	2 : (1,0)	1 : (0,0)	0 : (0,1)
3 : (1,1)	3 : (1,1)	2 : (1,0)	1 : (0,0)	0 : (0,1)
2 : (1,0)	2 : (1,0)	2 : (1,0)	1 : (0,0)	1 : (0,0)
1 : (0,0)	1 : (0,0)	1 : (0,0)	1 : (0,0)	1 : (0,0)
0 : (0,1)	0 : (0,1)	1 : (0,0)	1 : (0,0)	0 : (0,1)

Figure 2.14 *Conjonction pour une logique tétravalente codée bivalente*

La figure 2.14 modélise bien l'interprétation "vrai et purement formel", mais non pas l'interprétation "probabiliste" évoquée auparavant.

Nous pouvons aussi songer à passer à une sémantique des "mondes possibles". Supposons que les mondes possibles se réduisent à deux mondes : le monde actuel w_1 et le monde possible w_2. A toute proposition nous allons attribuer une des valeurs logiques 0, 1, 2 ou 3 selon qu'elle est :

- fausse dans w_1 et dans w_2 (nécessairement fausse);
- fausse dans w_1 et vraie dans w_2 (actuellement fausse mais pas nécessairement fausse);
- vraie dans w_1 et fausse dans w_2 (actuellement vraie mais pas nécessairement vraie);
- vraie dans w_1 et dans w_2 (nécessairement vraie).

Cette interprétation des valeurs logiques conduit aux tables de la figure 2.15 pour la négation, la conjonction et les opérateurs modaux.

Il est intéressant de noter que ce système, restreint aux opérateurs de négation et de conjonction, n'est autre que le produit direct de deux copies du système binaire $\{0, 1\}$. On vérifie ceci en représentant les valeurs logiques 0,1,2,3 par leur codage binaire :

$$0 = (0,0),\ 1 = (0,1),\ 2 = (1,0),\ 3 = (1,1).$$

Ce codage fournit la table de la figure 2.16 pour la conjonction.

Ce type de logique est compatible avec des univers constitués de k mondes possibles, notés $w_1, w_2, \cdots, w_k$. Les logiques comportent alors $m = 2^k$ valeurs de vérité.

$p \wedge q$

p	$\neg p$	$p \setminus q$	3	2	1	0	$\Box p$	$\Diamond p$
3	0	3	3	2	1	0	3	3
2	1	2	2	2	0	0	0	3
1	2	1	1	0	1	0	0	3
0	3	0	0	0	0	0	0	0

Figure 2.15 *Tables de vérité pour un univers à deux mondes*

$p \wedge q$

$p \setminus q$	3 : (1,1)	2 : (1,0)	1 : (0,1)	0 : (0,0)
3 : (1,1)	3 : (1,1)	2 : (1,0)	1 : (0,1)	0 : (0,0)
2 : (1,0)	2 : (1,0)	2 : (1,0)	0 : (0,0)	0 : (0,0)
1 : (0,1)	1 : (0,1)	0 : (0,0)	1 : (0,1)	0 : (0,0)
0 : (0,0)	0 : (0,0)	0 : (0,0)	0 : (0,0)	0 : (0,0)

Figure 2.16 *Table pour la conjonction*

Nous pouvons ainsi interpréter la classe des systèmes qui peuvent être représentés comme des produits cartésiens de sous-systèmes pour lesquels une interprétation existe déjà. Ces systèmes conduisent à la *sémantique des mondes possibles* dans l'approche *théorie du modèle.*

2.3.6. *Interprétation ensembliste de logiques multivaluées*

Des systèmes multivalués peuvent être construits en utilisant une définition ensembliste (ou *vériensembliste*) des valeurs de vérité. Considérons un ensemble W de mondes possibles : $W = \{w_1, w_2, \cdots, w_n\}$. Comme valeur de vérité d'une proposition nous allons prendre un certain *ensemble caractérisé sémantiquement*, à savoir le sous-ensemble de W constitué de tous les mondes possibles dans lesquels un certaine proposition p est vraie :

$V(p) = \{w | w \in W$ et p est vrai dans $w\}$.

La construction de ces systèmes est guidée par un principe sémantique. Ce principe conduit à la définition suivante de la sémantique des connecteurs (la notation "*" représente la complémentation ensembliste) :

$V(\neg p) = V(p^*) = W \setminus V(p)$
$V(p \wedge q) = V(p) \cap V(q)$
$V(p \vee q) = V(p) \cup V(q)$

$V(p \supset q) = V(\neg p \vee q)$
$V(p \equiv q) = V((p \supset q) \wedge (q \supset p))$

La structure des valeurs de vérité est une algèbre booléenne d'ensembles. Considérons par exemple un ensemble avec un seul monde, soit : $W = \{w\}$. Dans ce cas il n'y a que deux valeurs de vérité, soit :

$0 = \emptyset$ (ensemble vide) $= \mathbf{F}$
$1 = W = \{w\} = \mathbf{V}$

Les tables de vérité sont celles de l'algèbre booléenne classique. Considérons ensuite un ensemble avec trois mondes :

$7 = \{w_1, w_2, w_3\} = W = \mathbf{V}$
$6 = \{w_1, w_2\}$
$5 = \{w_1, w_3\}$
$4 = \{w_2, w_3\}$
$3 = \{w_3\}$
$2 = \{w_2\}$
$1 = \{w_1\}$
$0 = \emptyset = \mathbf{F}$

Dans cette approche, les modalités sont définies de la façon suivante :

$$\begin{array}{lll} V(\Box p) & = W = \mathbf{V} & \text{ssi} \quad V(p) = W \\ & = \emptyset = \mathbf{F} & \text{ssi} \quad V(p) \neq W \\ V(\Diamond p) & = W = \mathbf{V} & \text{ssi} \quad V(p) \neq \emptyset \\ & = \emptyset = \mathbf{F} & \text{ssi} \quad V(p) = \emptyset \end{array}$$

La notion de valeur distinguée est définie comme suit. Admettons qu'il existe un monde $w_r \in W$ qui est supposé être le monde réel. Tout sous-ensemble de mondes contenant le monde w_r est par définition une valeur distinguée.

Voici une variante de cette sémantique des mondes possibles. Admettons qu'il y ait un ensemble de propositions $p, q, \cdots$ qui soient des fonctions propositionnelles d'un paramètre numérique sur un domaine spécifié $\mathcal{D}$; soit par exemple $\mathcal{D} = \{0, 1, 2, \cdots\}$. Donc $p(x)$ pourrait être par exemple $x + 2 = 13$ ou $x - 3 \leq 5$. La valeur de vérité de x est par définition l'ensemble de toutes les valeurs de x où $p(x)$ est vrai :

$$V(p) = V(p(x)) = \{x | \in \mathcal{D} \text{ et } p(x) \text{ est vrai}\}.$$

Les valeurs de vérité des propositions sont donc à nouveau des *sous-ensembles d'un ensemble donné.*

La variable x peut enfin être interprétée comme une variable temporelle, ce qui revient à modéliser des événements temporels au moyen d'une sémantique ensembliste. Admettons que le temps soit divisé en n intervalles $\{I_1, \cdots, I_n\}$, où I_1 est interprété comme l'intervalle du temps présent. Soit des propositions $p, q, \cdots$ qui représentent des événements qui dépendent du temps, tels que, par exemple : "il pleut aujourd'hui" ou "la marquise sortit à cinq heures". Ces propositions sont représentées par des vecteurs de vérité à n places :

$<v_1, v_2, \cdots, v_n>$; $v_i \in \{1, 0\}$ pour tout i.

Les opérateurs modaux ont alors la signification suivante :

$\Box p$ $= \mathbf{V}$ si et seulement si p est vrai dans chaque intervalle;
$= \mathbf{F}$ dans les autres cas;
$\Diamond p$ $= \mathbf{V}$ si et seulement si p est vrai dans au moins un intervalle;
$= \mathbf{F}$ dans les autres cas.

Cette logique peut être modifiée afin de modéliser un système temporel où l'intervalle "présent" est situé à une place quelconque. Ceci permet de représenter des instants passés et des instants futurs.

2.3.7. *Structures modales et logiques multivaluées*

Etant donné une logique multivaluée, les opérateurs modaux peuvent être introduits comme des opérateurs propositionnels à une place définis au moyen de tables de vérité à une colonne, de la forme de la table de vérité pour la négation.

Par exemple, les trois opérateurs modaux de la logique aléthique traditionnelle sont les suivants :

$\Box p$ pour p est *nécessaire* (nécessairement vrai);
$\Diamond p$ pour p est *possible* (possiblement vrai);
$\bigcirc p$ pour p est *réel* (réellement vrai).

En logique à trois valeurs on s'attend à ce que les trois opérateurs modaux aient les tables de vérité indiquées dans la figure 2.17 :

	p	$\Box p$	$\Diamond p$	$\bigcirc p$
+V	+2	2	2	2
I	1	0	2	1
-F	-0	0	0	0

Figure 2.17 *Opérateurs modaux immergés dans une logique ternaire*

Soit D^+ et D^- respectivement les ensembles de valeurs distinguées et antidistinguées. On peut s'attendre à ce que les opérateurs modaux satisfassent aux règles suivantes :

($\Box 1$) Lorsque $V(p) \neq 2$ alors $V(\Box p) = 0$.
($\Box 2$) Lorsque $V(p) \notin D^+$ alors $V(\Box p) \notin D^+$.
($\Diamond 1$) Lorsque $V(p) \neq 0$ alors $V(\Diamond p) = 2$.
($\Diamond 2$) Lorsque $V(p) \notin D^-$ alors $V(\Diamond p) \in D^+$.
($\bigcirc 1$) $V(\bigcirc p) = 2$ si et seulement si $V(p) = 2$.
($\bigcirc 2$) $V(\bigcirc p) = 0$ si et seulement si $V(p) = 0$.
($\bigcirc 3$) $V(\bigcirc p) \in D^+$ si et seulement si $V(p) \in D^+$.
($\bigcirc 4$) $V(\bigcirc p) \in D^-$ si et seulement si $V(p) \in D^-$.

Les règles ($\bigcirc 1$) – ($\bigcirc 4$) peuvent toujours être satisfaites en imposant l'égalité : $V(\bigcirc p) = V(p)$.

Si la négation a comme propriété de projeter les valeurs de D^+ vers D^-, alors les règles ($\bigcirc 1$) – ($\bigcirc 4$) satisfont la propriété : $V(\bigcirc p) = V(\neg \bigcirc \neg p)$.

De plus les règles ($\Box 1$), ($\Box 2$) et ($\Diamond 1$), ($\Diamond 2$) peuvent alors être liées par les relations : $V(\Box p) = V(\neg \Diamond \neg p)$ et $V(\Diamond p) = V(\neg \Box \neg p)$.

Etant donné les modalités définies par les tables de vérité d'une logique à plusieurs valeurs, ainsi que la sémantique habituelle des connecteurs propositionnels, on peut se poser la question suivante : Comment les modalités des systèmes multivalués peuvent-ils être axiomatisés ? Les systèmes les mieux connus sont ceux construits par Lewis entre les années 1918 et 1932. Au nombre de 5, ils sont représentés par les sigles $S1$ – $S5$, $S1$ étant le système le plus faible et $S5$ le plus fort [Rescher 69].

p	$\neg p$	$\Box p$	$\Diamond p$
+3	0	3	3
2	1	0	3
1	2	0	3
0	3	0	0

$p \wedge q$

$p \setminus q$	3	2	1	0
3	3	2	1	0
2	2	2	0	0
1	1	0	1	0
0	0	0	0	0

$p -\!\!< q$

$p \setminus q$	3	2	1	0
3	3	0	0	0
2	3	3	0	0
1	3	0	3	0
0	3	3	3	3

$p >\!\!- q$

$p \setminus q$	3	2	1	0
3	3	0	0	0
2	0	3	0	0
1	0	0	3	0
0	0	0	0	3

Figure 2.18 *Tables associées au système $S5$*

On peut montrer aisément qu'il existe des logiques à plusieurs valeurs qui incluent les théorèmes de ces systèmes, en ce sens que chaque théorème du système est une tautologie de la table de vérité de la logique multivaluée. Par exemple, les tables de vérité de la figure 2.18 incluent les théorèmes de $S5$

et donc ceux de tous les autres systèmes de Lewis. La figure 5.19 décrit les systèmes axiomatiques des logiques $S1$-$S5$.

Primitives :	$\neg$, $\wedge$, $\Box$
Définitions :	$\Diamond p =_{def} \neg\Box\neg p$
	$p \vee q =_{def} \neg(\neg p \wedge \neg q)$
	$p \supset q =_{def} \neg(p \wedge \neg q)$
	$p \equiv q =_{def} (p \supset q) \wedge (q \supset p)$
	$p \prec q =_{def} \Box(p \supset q)$ (implication stricte)
	$p \succ\!\prec q =_{def} \Box(p \equiv q)$ (équivalence stricte)
Règles d'inférence :	*Substitution uniforme*
	Modus ponens : $\frac{\vdash p, \vdash (p \prec q)}{\vdash q}$
	Conjonction : $\frac{\vdash p, \vdash q}{\vdash p \wedge q}$

Axiomes pour $S1$:	(1) $(\alpha \wedge \beta) \prec (\beta \wedge \alpha)$
	(2) $(\alpha \wedge \beta) \prec \alpha$
	(3) $\alpha \prec (\alpha \wedge \alpha)$
	(4) $(\alpha \wedge (\beta \wedge \gamma)) \prec (\beta \wedge (\alpha \wedge \gamma))$
	(5) $((\alpha \prec \beta) \wedge (\beta \prec \gamma)) \prec (\alpha \prec \gamma)$
	(6) $\alpha \prec \Diamond\alpha$

Axiomes pour $S2$:	(1)–(6) plus :
	(7.2) $\Diamond(\alpha \wedge \beta) \prec \Diamond\alpha$

Axiomes pour $S3$:	(1)–(6) plus :
	(7.3) $(\alpha \prec \beta) \prec (\Diamond\alpha \prec \Diamond\beta)$

Axiomes pour $S4$:	(1)–(6) plus :
	(7.4) $\Diamond\Diamond\alpha \prec \Diamond\alpha$

Axiomes pour $S5$:	(1)–(6) plus :
	(7.5) $\Diamond\alpha \prec \Box\Diamond\alpha$

Figure 2.19 *Systèmes logiques de Lewis*

Chapitre 3
Logiques modales temporelle et épistémique

3.1. Le temps et le changement

3.1.1. *Introduction*

Les objets, les êtres, les relations, sont affectés par l'écoulement du temps. Les techniques de l'intelligence artificielle doivent donc être capables de raisonner sur des données, des connaissances, qui sont susceptibles de se modifier au cours du temps. Ces techniques seront principalement d'application dans les contextes des bases de données temporelles, des systèmes experts et du traitement de la langue naturelle.

Les bases de données temporelles sont celles où les données se modifient avec le temps. Le modèle proposé par Codd [Codd 70] considère une base de données comme une collection de relations variables. Plus récemment, Clifford a introduit le concept de *base de données historiques* comme outil de modélisation pour les changements d'états du monde [Clifford 90].

Certains systèmes experts utilisent des raisonnements qui font intervenir l'écoulement du temps [Jackson et al. 89]. Un système expert médical devra, par exemple, gérer des informations du type : *Prendre le médicament x jusqu'à ce que le symptôme y disparaisse; ensuite commencer le traitement z durant un mois; si le symptôme y réapparaît durant le suivi de ce traitement z, reprendre le médicament x sans interrompre le traitement z.*

Enfin, les discours en langue naturelle utilisent normalement des phrases comportant un aspect temporel. Que l'on songe aux différentes conjugaisons des verbes : le présent, le passé et le futur de l'indicatif servent à décrire les moments où des actions se produisent. Le *traitement de la langue naturelle*, c'est-à-dire l'analyse formalisée des phrases de la langue suivie de leur

traduction dans une langue artificielle, impose à cette dernière de comporter également une composante temporelle.

3.1.2. *Relations entre temps et changement*

Que ce soit dans le cadre des bases de données, des systèmes experts, ou du traitement de la langue naturelle, il s'agit de prendre en compte le fait que l'écoulement du temps est marqué par la modification des objets et des êtres à propos desquels on raisonne, et qui font partie de ce que l'on a coutume d'appeler *le domaine du discours.* Le concept de *temps* est donc intimement lié à celui de modification, de *changement.*

Nous pouvons envisager plusieurs types de relations entre temps et changement.

1. **Le temps préexiste au changement**
   C'est le *modèle newtonien* où le temps est pris comme variable indépendante. Dans ce modèle :
   (a) On se donne des instants $t_1, t_2, \cdots, t_n$, et une relation entre instants, notée $<$; $t_1 < t_2$ signifie que l'instant t_1 précède l'instant t_2.
   (b) A chaque événement est associé un temps.
   (c) L'événement e_1 précède l'événement e_2 si le temps de e_1 précède celui de e_2.

2. **Le changement préexiste au temps**
   C'est le *modèle leibnizien* où le changement est pris comme variable indépendante. Dans ce modèle :
   (a) On se donne des événements $e_1, e_2, \cdots, e_m$, et un ensemble de relations qui les lie, telles la simultanéité, la succession, la causalité.
   (b) Les instants temporels sont identifiés en termes de ces relations, par exemple comme des classes d'équivalence sous la relation de simultanéité. Dans ce cas, t_1 précède t_2 si chaque événement au temps t_1 précède chaque événement au temps t_2.

Les modèles où le temps dépend des événements sont en général considérés comme plus conformes à la réalité, plus adéquats au réel, étant donné que ce que l'on observe, ce sont les occurrences d'événements, les modifications d'état, non le temps qui s'écoule.

3.1.3. *Le temps : ensemble de points ou d'intervalles*

Si l'on peut choisir de définir le changement en fonction du temps, ou bien le temps en fonction du changement, le temps lui-même peut encore être caractérisé au moyen de deux types de primitives : l'*instant* ou l'*intervalle.*

3.1.3.1. *Les instants comme primitives du temps*

On peut tout d'abord adopter les *instants* comme primitives : ce sont les points, ou *atomes* du temps. Ils peuvent avoir ou ne pas avoir de *durée*. S'ils n'en ont pas, chaque intervalle fini comporte une infinité d'instants. Le temps est, dans ce cas, *dense* ou *continu*. Si les instants ont une durée, le temps est discrétisé; on parle alors de *temps newtonien*. Dans une structure *discrète*, la *durée* est le *nombre* d'instants entre t_1 et t_2.

Les *intervalles* sont ensuite définis comme des paires ordonnées d'instants. Si t_1 et t_2 sont des instants, et que $t_1 < t_2$, alors (t_1, t_2) est un intervalle. On peut également définir un intervalle comme un ensemble convexe d'instants : $\{t | t_1 < t < t_2\}$.

Les relations entre intervalles sont définis au moyen de la précédence temporelle, notée : "$<$" (voir [Galton 95]) :

1. (t, u) est *avant* (v, w) si $u < v$;
2. (t, u) *rencontre* (v, w) si $u = v$;
3. (t, u) *coïncide partiellement* avec (v, w) si $t < v < u < w$;
4. (t, u) *commence* (v, w) si $t = v$ et $u < w$;
5. (t, u) est *inséré dans* (v, w) si $v < t$ et $u < w$;
6. (t, u) *termine* (v, w) si $v < t$ et $u = w$;
7. (t, u) *égale* (v, w) si $t = v$ et $u = w$.

3.1.3.2. *Les intervalles comme primitives du temps*

On peut adopter des *intervalles* comme éléments primitifs du temps. Dans ce cas, on dit que l'intervalle t_1 précède l'intervalle t_2, ce qui s'écrit : $(t_1 < t_2)$, si t_1 précède *entièrement* t_2. On définit ainsi un ordre partiel sur l'ensemble des intervalles.

Les prédécesseurs et les successeurs d'un intervalle t sont respectivement définis comme suit :

$pred(t) = \{x | x < t\}$;
$succ(t) = \{x | t < x\}$.

Les relations entre intervalles sont définies au moyen des deux fonctions *pred* et *succ* :

1. t_1 précède t_2 si $succ(t_1) \cap pred(t_2) \neq \emptyset$;
2. t_1 rencontre t_2 si $t_1 < t_2$ et $succ(t_1) \cap pred(t_2) = \emptyset$;
3. t_1 recouvre t_2 si $pred(t_1) \subset pred(t_2)$, $succ(t_2) \subset succ(t_1)$, et $t_1 \not< t_2$;
4. t_1 commence t_2 si $pred(t_1) = pred(t_2)$ et $succ(t_1) \subset succ(t_2)$;
5. t_1 est inclus dans t_2 si $pred(t_1) \subset pred(t_2)$ et $succ(t_1) \subset succ(t_2)$;
6. t_1 termine t_2 si $pred(t_2) \subset pred(t_1)$ et $succ(t_1) = succ(t_2)$;

7. t_1 égale t_2 si $pred(t_1) = pred(t_2)$ et $succ(t_1) = succ(t_2)$.

L'*instant* est ensuite défini comme la *rencontre* de deux intervalles. Pour ce faire, on caractérise un intervalle t comme *initial* s'il n'est précédé par aucun autre, et comme *final* s'il n'est suivi d'aucun autre; les fonctions $\delta^-(t)$ et $\delta^+(t)$ sont définies comme suit :

$$\delta^-(t) = \begin{cases} -\infty & \text{si } t \text{ est initial;} \\ \{(x,y) | x \text{ rencontre } y \text{ et } x \text{ rencontre } t\} & \text{autrement.} \end{cases}$$

$$\delta^+(t) = \begin{cases} \infty & \text{si } t \text{ est final;} \\ \{(x,y) | x \text{ rencontre } y \text{ et } t \text{ rencontre } y\} & \text{autrement.} \end{cases}$$

De ces définitions, on déduit que si t_1 et t_2 commencent au même moment, alors $\delta^-(t_1) = \delta^-(t_2)$, tandis que s'ils terminent au même moment, alors $\delta^+(t_1) = \delta^+(t_2)$; de plus, t_1 rencontre t_2 si et seulement si $\delta^+(t_1) = \delta^-(t_2)$. Nous pouvons donc définir $\delta^-(t)$ et $\delta^+(t)$ comme les instants qui limitent l'intervalle t, respectivement à son début et à sa fin.

Nous avons ainsi décrit une façon de construire des instants à partir d'intervalles, aussi bien que des intervalles à partir d'instants.

3.1.4. *Etats et événements*

Le temps ou le changement peuvent être adoptés comme primitives. Le changement, qu'il soit considéré ou non comme primitive, peut lui-même être interprété en terme d'*états*. Le changement est perceptible par la *modification* d'état, par le *passage* d'un état à un autre, ou par la *découverte* d'un nouvel état.

Une façon de procéder est d'identifier chaque instant à un état du monde. Si donc on identifie l'état s_1 au temps t_1 et l'état s_2 au temps t_2, et si $s_1 \neq s_2$ et $t_1 < t_2$, alors le monde change de l'état s_1 vers l'état s_2 entre les temps t_1 et t_2. Le changement est donc la conséquence de l'assignation d'états différents aux différents temps.

Les modèles de ce type, où le changement résulte de l'assignation d'états au temps, sont appelés modèles *basés sur l'état*. Il ne s'agit pas nécessairement de modèles où le temps est indépendant. Les états du monde peuvent être définis en premier, avec une relation d'ordre imposée directement sur l'ensemble des états du monde, et les temps introduits ultérieurement. On attribue ensuite des indices numériques aux états en conformité avec leur ordre temporel intrinsèque.

Comme le temps, les *événements* peuvent également être définis en termes d'états. Considérons par exemple l'événement de mourir. Pour mourir il faut passer de l'état de vivant à celui de mort. Mais intuitivement l'état d'être mort peut uniquement être défini comme l'événement de mourir appliqué à l'état de vivant.

Donc, une façon de rendre compte d'un changement est de postuler la notion d'événement, qui est distincte et indépendante de celle d'état. On peut décider de ne reconnaître comme états que ceux qui peuvent être définis en termes d'événements. On parle dans ce cas d'une notion de changement *basée sur l'événement.* Dans un tel modèle, on doit spécifier les relations causales et logiques entre états et événements. Par exemple, l'événement de se marier est relié causalement aux états de célibataire et de marié.

Nous voyons donc que nous sommes en présence d'un paysage bien plus vaste que celui que l'on trouve dans le paradigme newtonien. Lorsque nous considérons des systèmes où intervient le temps, nous sommes amenés à choisir entre toute une série d'options. A une extrémité nous avons l'environnement newtonien lui-même, avec un ensemble d'instants qui peuvent chacun être associés à un état du monde. Le changement est alors simplement le résultat de la définition de plusieurs états et du passage d'un état vers un autre. A l'autre extrémité, nous avons l'environnement où les primitives sont les événements reliés entre eux par des relations de simultanéité et de succession. Le temps intervient dans cet environnement uniquement par la notion de temps d'*un* événement, typiquement un intervalle; les temps ainsi construits héritent leur ordre de celui des relations entre événements.

3.2. La logique temporelle des propositions

3.2.1. *Syntaxe*

Le langage de la logique temporelle considéré dans ce paragraphe est un langage propositionnel auquel on ajoute deux opérateurs modaux universels, à savoir $[F]$ et $[P]$, dont les significations respectives sont "toujours dans le futur" et "toujours dans le passé". Les opérateurs existentiels correspondants, $<F>$ et $<P>$, signifient respectivement "parfois dans le futur" et "parfois dans le passé". Les deux types d'opérateurs satisfont les relations de dualité des opérateurs modaux :

$[F]=\neg<F>\neg$;
$[P]=\neg<P>\neg$.

Ces opérateurs nous permettent de construire des expressions logiques temporelles intéressantes du point de vue de l'expressivité; voici certaines de ces expressions ainsi que leur interprétation dans la langue naturelle :

$[F]<F>A$	:	A deviendra toujours vrai ultérieurement.
$<P>[P]A$	:	au moins une fois, A avait toujours été vrai.
$[F](A \supset <F>B)$	:	A rendra toujours B vrai dans un futur.

Les opérateurs temporels permettent de transposer en logique les conjugaisons de la langue naturelle.

3.2.2. *Sémantique*

Une structure pour ce langage a la forme $< W, R_F, R_P >$. Si $\mathcal{M}$ est un modèle, nous avons :

$$\mathcal{M} \models_s [F]A \text{ ssi } sR_Ft \text{ implique } \mathcal{M} \models_t A \text{ pour tout } t \in W;$$
$$\mathcal{M} \models_s [P]A \text{ ssi } sR_Pt \text{ implique } \mathcal{M} \models_t A \text{ pour tout } t \in W.$$

Nous interprétons les propriétés sR_Ft et sR_Pt respectivement comme "t est dans le futur de s" et "t est dans le passé de s". Nous exigeons évidemment que t soit dans le passé de s si et seulement si s est dans le futur de t, c'est-à-dire :

$$sR_Pt \Longleftrightarrow tR_Fs.$$

(Les relations R_P et R_F sont inverses l'une de l'autre.)

Etant donné une structure $\mathcal{F} =< W, R_F, R_p >$, on peut démontrer les équivalences suivantes :

- $\mathcal{F} \models A \supset [P]<F>A \Longleftrightarrow \forall s \forall t\, (sR_Pt \Rightarrow tR_Fs)$.
- $\mathcal{F} \models A \supset [F]<P>A \Longleftrightarrow \forall s \forall t\, (sR_Ft \Rightarrow sR_Pt)$.
- Si une logique normale contient le schéma :

  $$A \supset [P]<F>A,$$

  alors les relations R_P et R_F possèdent la propriété :

  $$sR_Pt \Rightarrow tR_Fs.$$

- Si une logique normale Λ contient le schéma :

  $$A \supset [F]<P>A,$$

  alors les relations possèdent la propriété :

  $$tR_Fs \Rightarrow sR_Pt.$$

Ces propositions formalisent les faits suivants.

- Si une formule A est vraie actuellement, alors il existe un passé pour lequel A deviendra vraie dans le futur.
- Si une formule A est vraie actuellement, alors il existe un futur pour lequel A aura été vraie dans le passé.

Comme nous l'avons indiqué, les relations R_F et R_P peuvent être définies l'une à partir de l'autre; nous pouvons donc également choisir une seule de ces relations comme primitive et utiliser des structures $\mathcal{F} =< W, R >$, avec $R \subset W \times W$:

$$\mathcal{M} \models_s [F]A \Longleftrightarrow (sRt \Rightarrow \mathcal{M} \models_t A);$$
$$\mathcal{M} \models_s [P]A \Longleftrightarrow (tRs \Rightarrow \mathcal{M} \models_t A).$$

Le modèle temporel ainsi défini est absolument général et est basé uniquement sur la relation de précédence temporelle. Le temps physique satisfait cependant à des contraintes supplémentaires, qui rendent par exemple compte de diverses propriétés mathématiques de sa relation d'ordre. Les plus simples de ces contraintes peuvent être exprimées en termes de la logique des prédicats du premier ordre :

transitivité : $\forall x \forall y \forall z((x < y \wedge y < z) \supset x < z)$
non-réflexivité : $\forall x \neg(x < x)$
linéarité : $\forall x \forall y(x < y \vee y < x \vee x = y)$

D'autres contraintes sont typiquement du *second ordre*, c'est-à-dire incluant des quantifications sur des ensembles de points :

Continuité : chaque ensemble borné supérieurement a une plus petite borne supérieure.
Homogénéité : chaque point peut être projeté sur un autre par un isomorphisme d'ordre sur la structure temporelle.

Les contraintes sur la structure temporelle seront examinées plus en détail ultérieurement (paragraphe 3.2.6).

3.2.3. *Axiomatique*

On obtient une *logique temporelle normale* en n'imposant aucune restriction à la relation R. Un système axiomatique pour une logique temporelle normale est constitué des trois ingrédients suivants :

- Un système axiomatique de la logique propositionnelle.
- Les schémas d'axiome :

K_F : $[F](A \supset B) \supset ([F]A \supset [F]B)$;
K_P : $[P](A \supset B) \supset ([P]A \supset [P]B)$;
C_F : $A \supset [F]<P>A$;
C_P : $A \supset [P]<F>A$.

- Les règles d'inférence de nécessité :

$\vdash A \Rightarrow \vdash [F]A$;
$\vdash A \Rightarrow \vdash [P]A$.

Une logique temporelle normale est donc une logique normale dans le langage des opérateurs $[F]$ et $[P]$. La plus petite logique temporelle normale est notée K_t. Toutes les autres logiques temporelles sont obtenues à partir d'une logique

normale et de certaines conditions imposées à la relation R. Ces propriétés déterminent le choix des schémas qu'il faut ajouter au système axiomatique normal.

Pour toutes les formules de la logique temporelle K_t, nous pouvons démontrer que :

$\vdash_{K_t} A$ si et seulement si $\models A$.

3.2.4. *Logique temporelle linéaire*

La logique temporelle linéaire permet de décrire des séquences de situations et de raisonner à leur sujet. On peut interpréter ces séquences de situations de plusieurs manières. Dans une interprétation proprement temporelle, la séquence de situations représente l'évolution de l'état du monde au cours du temps. Une autre interprétation concerne le cas où la séquence de situations représente les états successifs d'un programme en cours d'exécution. Cette interprétation permet d'utiliser la logique temporelle pour la spécification et la vérification de programmes [Gribomont 85].

Il existe de nombreuses versions de la logique temporelle linéaire. Nous en présentons une, souvent utilisée dans la spécification et la vérification de programmes parallèles.

Le langage de la logique temporelle linéaire est celui de la logique temporelle enrichi de trois symboles désignant des *opérateurs temporels spécialisés*. Ces symboles sont les suivants :

1. $\bigcirc$ qui se lit "à l'instant suivant";
2. $\mathcal{U}$ qui se lit "jusqu'à" (la notation $\mathcal{U}$ vient du mot anglais "until");
3. $\mathcal{S}$ qui se lit "depuis" (la notation $\mathcal{S}$ vient du mot anglais "since").

L'opérateur $\bigcirc$ est unaire tandis que les opérateurs $\mathcal{U}$ et $\mathcal{S}$ sont binaires. La sémantique de ces opérateurs est la suivante. La formule $\bigcirc A$ a la même interprétation au temps t que la formule A au temps $t+1$. La formule $A\,\mathcal{U}\,B$ est vraie dans deux cas : quand A est toujours vraie, ou quand B devient vraie (sans nécessairement le rester), au plus tard à l'instant où A devient fausse. L'opérateur $\mathcal{U}$ ainsi défini est souvent appelé "until faible". Une variante fréquente de l'opérateur $\mathcal{U}$, l'"until fort", exclut l'option où A est toujours vraie. Enfin, l'opérateur $\mathcal{S}$, *depuis*, est le dual de $\mathcal{U}$. Les définitions sémantiques sont les suivantes :

- $\mathcal{M} \models_t \bigcirc A$ si et seulement si $\mathcal{M} \models_{t+1} A$, où le point $t+1$ est le successeur immédiat de t dans la séquence.
- $\mathcal{M} \models_t A\,\mathcal{U}\,B$ si et seulement si pour un certain $t < t'$, $\mathcal{M} \models_{t'} A$, et pour tout t'' entre t' et t, $\mathcal{M} \models_{t''} B$.
- $\mathcal{M} \models_t A\mathcal{S}B$ si et seulement si pour un certain $t' < t$, $\mathcal{M} \models_{t'} A$, et pour tout t'' entre t et t', $\mathcal{M} \models_{t''} B$.

3.2.4.1. *Exemples*

1. La formule p est vérifiée par toute séquence dont le premier état vérifie p.
2. La formule $\Box(p \supset \bigcirc q)$ est vérifiée par toute séquence dans laquelle chaque état où p est vrai est immédiatement suivi d'un état où q est vrai.
3. La formule $\Box(p \supset \bigcirc(\neg q \,\mathcal{U}\, r))$ est vérifiée par toute séquence telle que si p est vrai dans un état, alors q reste faux à partir de l'état suivant et jusqu'au premier état où r est vrai.
4. La formule $\Box\Diamond p$ est vérifiée par toute séquence où p est vrai infiniment souvent.
5. La formule $\Box p \wedge \Diamond\neg p$ n'est vérifiée par aucune séquence.
6. La formule $\neg p \,\mathcal{U}\, p$ est vérifiée par toute séquence (elle est donc valide).

3.2.4.2. *Axiomatisation*

Pour axiomatiser la logique temporelle linéaire que nous venons de définir, nous pouvons partir de l'axiomatisation de la logique modale, donnée au paragraphe 2.2.6. Notre système axiomatique comporte donc d'abord :

(A0) toutes les instances de schémas valides de la logique propositionnelle.

Voyons ensuite comment axiomatiser l'opérateur $\bigcirc$. La logique temporelle est normale (§ 2.2.6) et l'opérateur $\bigcirc$ est fonctionnel. Nous avons donc le schéma d'axiome K ainsi que l'axiome caractérisant l'existence et l'unicité du successeur :

(A1) $\bigcirc(A \supset B) \supset (\bigcirc A \supset \bigcirc B)$;

(A2) $\neg\bigcirc A \equiv \bigcirc\neg A$.

Examinons à présent les opérateurs $\Box$ et $\Diamond$. Ceux-ci satisfont à l'axiome de dualité et à l'axiome K :

(A3) $\Diamond A \equiv \neg\Box\neg A$;

(A4) $\Box(A \supset B) \supset (\Box A \supset \Box B)$.

Plutôt que de donner les axiomes caractérisant les propriétés de l'opérateur $\Box$ (réflexivité, transitivité, linéarité, etc.) il est plus commode de donner les axiomes établissant le lien entre les modalités $\Box$ et $\bigcirc$, c'est-à-dire :

(A5) $\Box A \supset (A \wedge \bigcirc A \wedge \bigcirc\Box A)$;

(A6) $\Box(A \supset \bigcirc A) \supset (A \supset \Box A)$.

Enfin, l'opérateur $\mathcal{U}$ est caractérisé par les deux axiomes suivants :

(A7) $A \,\mathcal{U}\, B \supset (B \vee (A \wedge \bigcirc(A \,\mathcal{U}\, B)))$;

(A8) $[X \wedge \Box(X \supset (B \vee (A \wedge \bigcirc X)))] \supset A \,\mathcal{U}\, B$.

Dans ce dernier axiome (qui, comme tous les autres, est en réalité un schéma d'axiome), X désigne une formule arbitraire. Pour compléter notre système axiomatique, nous devons encore considérer les règles d'inférence. Comme en logique modale, celles-ci sont le *modus ponens* et la *règle de nécessité* :

(*MP*) si $\vdash A$ et si $\vdash A \supset B$, alors $\vdash B$;

(*RN*) si $\vdash A$, alors $\vdash \Box A$.

Le système axiomatique de la logique temporelle propositionnelle linéaire est adéquat et complet.

3.2.5. *Opérateurs spécialisés*

Les opérateurs de la logique temporelle sont ceux de la logique modale. Les particularités de certains modèles temporels rendent utiles l'introduction de nouveaux opérateurs qui vont faciliter l'usage des langages, mais non pas en accroître la puissance. Nous avons rencontré jusqu'à présent trois opérateurs de ce type pour la logique temporelle linéaire; en voici quelques autres.

L'opérateur de *différence*, noté $\mathcal{D}$, est défini comme suit : la formule $\mathcal{D}A$ est vraie dans un certain monde si et seulement si la formule A est vraie dans un *autre* monde; ce qui s'écrit :

$$\mathcal{M} \models_t \mathcal{D}A \quad \text{si et seulement si il existe un certain } t' \neq t \text{ pour lequel } \mathcal{M} \models_{t'} A.$$

Cette définition permet de résoudre certaines limitations du formalisme de base. Par exemple, moyennant une modification de la théorie du modèle, la *non-réflexivité* peut être définie au moyen du principe [van Benthem 95] :

$$<F>A \supset \mathcal{D}\,A.$$

La logique minimale K_t peut ensuite être augmentée au moyen d'axiomes qui rendent la relation associée aussi proche que possible de ce que l'on souhaite modéliser par la locution : "dans un autre monde".

$\mathcal{D} \neg \mathcal{D} \neg A \supset A$	:	symétrie
$\mathcal{D}\ \mathcal{D}\,A \supset (\mathcal{D}\,A \vee A)$	:	pseudo-transitivité
$<F>A \supset (\mathcal{D}\,A \vee A)$	:	$\forall x \forall y (x < y \supset (xR_D y \vee x = y))$

Lorsque l'on souhaite poursuivre l'axiomatisation de la logique sur cette base, on est amené à définir une nouvelle loi d'inférence :

si $\vdash (p \wedge \neg \mathcal{D}\,p) \supset A$ (p n'apparaît pas dans A), alors $\vdash A$.

Sémantiquement, cette loi exprime la *non-réflexivité* de la relation R_D, ce qui n'était pas exprimable dans le langage de base de la logique modale (voir [de Rijke 93]).

Dans la langue naturelle, il existe d'autres temps que le passé et le futur. Par exemple, la locution "étant" sous-entend une certaine durée dont la signification

peut être rendue par un opérateur modal, noté Π :

$\mathcal{M} \models_t \Pi A$ si et seulement si il existe des temps t_1 et t_2, avec $t_1 < t$ et $t < t_2$ tels que pour tout t' entre t_1 et t_2, $\mathcal{M} \models_{t'} A$.

3.2.6. *Propriétés additionnelles de la relation d'accessibilité*

Une relation R est *linéaire à gauche* si :

$$\forall s \forall t \forall u \, ((tRu \wedge sRu) \supset tRs \vee t = s \vee sRt).$$

Cette propriété signifie que si t précède u et si s précède u, alors t précède s, ou bien t est identique à s, ou bien s précède t. Le concept de "temps arborescent" ou "ramifié" ("branching time" en anglais) est obtenu en imposant à la relation R des conditions de linéarité à gauche et de transitivité.

Considérons une séquence d'événements E_1, E_2, E_4; on peut admettre que cette séquence est une succession d'événements parmi un ensemble d'autres séquences (physiquement) possibles; par exemple la séquence E_1, E_3, E_5. On peut représenter cet ensemble de séquences au moyen du diagramme de la figure 3.1. L'idée de base du "temps arborescent" est que chaque événement E_i est précédé immédiatement d'un seul événement (le passé est unique) mais peut être suivi immédiatement de plusieurs événements (le futur est multiple).

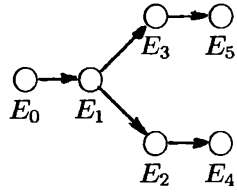


Figure 3.1 *Séquences dans une structure temporelle arborescente*

La propriété de *transitivité* imposée à la relation R entraîne l'introduction des deux schémas suivants dans le système axiomatique de la logique temporelle :

$[F]A \supset [F][F]A$;

$[P]A \supset [P][P]A$.

La propriété de linéarité à gauche se traduit par le schéma d'axiome suivant [Cochiarella 66] :

$$([P](A \vee B) \wedge [P](A \vee [P]B) \wedge [P]([P]A] \vee B)) \supset ([P]A \vee [P]B).$$

Une transformation de ce schéma en termes de l'opérateur existentiel $<P>$ fournit le schéma équivalent :

$$(<P>A \wedge <P>B) \supset (<P>(A \wedge B) \vee <P>(A \wedge <P>B) \vee <P>(B \wedge <P>A)).$$

Ce schéma indique que si un événement A et un événement B se produisent dans le passé, alors, ou bien A et B se produisent au même moment dans le passé, ou bien B est antérieur à A dans le passé, ou bien A est antérieur à B dans le passé.

On obtient ainsi un système axiomatique pour une logique temporelle qui formalise le concept de "temps arborescent" en ajoutant trois schémas d'axiomes au système axiomatique de la logique temporelle minimale K_t. La logique ainsi obtenue est notée K_b.

La structure temporelle arborescente est caractérisée par la présence de deux ordres; un ordre total ($<$) sur les instants et un ordre partiel ($\prec$) sur les événements. Ces ordres sont tels que :

$$E_1 \prec E_2 \Longrightarrow t(E_1) < t(E_2).$$

La structure possède les propriétés de *transitivité* (si $E_0 \prec E_1$ et si $E_1 \prec E_2$, alors $E_0 \prec E_2$) et de *linéarité à gauche*. Le temps arborescent sert à modéliser des *alternatives historiques possibles*.

Caractérisés par des futurs uniques et des passés multiples, les modèles temporels *linéaires à droite* formalisent des phénomènes observés (connus) mais dont l'histoire est partiellement inconnue. Ce sont par exemple les phénomènes cosmologiques. A partir des observations actuelles, trois modèles cosmologiques (au moins) sont concevables : l'univers statique (création continue), l'univers pulsatoire, le modèle du "big bang". Un autre exemple concerne les événements historiques. A partir de manuscrits latins et grecs des évangiles que nous possédons actuellement, il est possible d'imaginer des rédactions originelles soit en grec, soit en hébreu, soit en araméen.

Dans la plupart des applications (physique et mécanique classiques), le temps est linéaire : non seulement le passé est unique (comme dans le cas du "temps arborescent") mais aussi le futur est unique; dès lors la relation R doit posséder la propriété de *linéarité* :

$$\forall s \forall t\ (sRt \vee s = t \vee tRs).$$

Un système axiomatique pour un temps linéaire s'obtient donc à partir du système axiomatique pour la logique K_b en y ajoutant le schéma d'axiome :

$$([F](A \vee B) \wedge [F](A \vee [F]B) \wedge [F]([F]A \vee B)) \supset ([F]A \vee [F]B).$$

La logique ainsi obtenue est notée K_l.

La définition de la logique K_l laisse en suspens de nombreuses questions fondamentales que l'on peut se poser à propos du temps. Le temps a-t-il un commencement et une fin ? Existe-t-il un moment entre deux moments distincts ? Le temps a-t-il une structure continue comme les nombres réels ?

Tout d'abord, si le temps n'a ni début ni fin, alors il doit posséder la propriété de *reproductivité*, c'est-à-dire :

$$\forall s \exists t\ (sRt) \text{ et } \forall s \exists t\ (tRs).$$

L'extension de la logique temporelle K_l qui inclut cette propriété est notée K_s. On obtient une axiomatisation de K_s en ajoutant aux schémas de K_l les deux schémas suivants :

$[F]A \supset <F>A$;

$[P]A \supset <P>A$.

Existe-t-il un moment entre deux moments ? Si l'on répond à cette question par l'affirmative on en déduit une conception de la structure du temps qui est semblable à celle des nombres rationnels; on dit que le temps est "dense". La propriété de *densité* est représentée par la formule :

$$\forall s \forall t\,(sRt \supset \exists u(sRu \wedge uRt)).$$

La densité du temps s'exprime par les schémas d'axiomes :

$[F][F]A \supset [F]A$;

$[P][P]A \supset [P]A$.

On peut montrer la liaison existant entre la notion de densité et le premier de ces schémas d'axiomes; nous allons raisonner sur la formule $<F>A \supset <F><F>A$, qui est équivalente au premier schéma. Si $<F>A$ est vraie en t, il y a un moment t' dans le futur de t où A est vraie. De plus, si le temps est dense, il y a un moment t'' situé entre t et t' où $<F>A$ est vraie, et donc $<F><F>A$ est vraie en t. La logique du temps dense se note K_{ld}.

En opposition à la propriété de densité, on peut imposer une *discrétisation du temps*, de telle sorte que l'ordre des moments temporels soit isomorphe à celui des nombres entiers. Cette discrétisation du temps implique l'introduction dans le système des schémas d'axiome :

D_F : $<F>\top$
D_P : $<P>\top$
Z_F : $[F]([F]A \supset A) \supset (<F>[F]A \supset [F]A)$
Z_P : $[P]([P]A \supset A) \supset (<P>[P]A \supset [P]A)$

3.2.7. *Logique temporelle non transitive*

Il est possible de développer une logique temporelle cohérente où le temps n'est *pas* transitif. Supposons que les instants temporels soient les *nombres réels*. La relation d'accessibilité, notée $\lhd$, signifie :

$t \lhd t'$: t précède t' de *moins* d'une heure.

Cette relation n'est pas transitive; en effet :

$0 \lhd 0.6$ car $(0.6) - 0 < 1$;
$0.6 \lhd 1.2$ car $(1.2) - (0.6) < 1$;
$0 \not\lhd 1.2$ car $(1.2) - (0) > 1$.

Dans ce type de schéma, les formules modales sont interprétées comme suit :

$<P>A$: A a été vérifié au cours de l'heure passée;
$[P]A$: A a été vérifié jusqu'à l'heure passée;
$<F>A$: A sera vérifié au cours de l'heure qui vient;
$[F]A$: A sera vérifié à partir de l'heure qui vient;

A partir de ces interprétations, on déduit la signification des formules suivantes :

$<P><P>A$: A a été vérifié au cours des deux heures passées;
$[F][F][F]A$: A sera vérifié pendant les trois heures qui viennent;
$<P><P>A \wedge \neg <P>A$: A a été vérifié au cours des deux heures passées, mais pas lors de la dernière.

3.2.8. *Logique temporelle à plusieurs dimensions*

D'un point de vue calculatoire, il n'y a aucun inconvénient à considérer des formules à plusieurs variables, et d'interpréter chacune de ces variables comme représentant une grandeur temporelle. L'évaluation de ces formules se fait alors selon un vecteur temporel, soit :

$$\mathcal{M} \models_{<t_1,\ldots,t_n>} A.$$

Il existe plusieurs applications où ce type de formule a un sens. Par exemple, dans l'étude linguistique des temps, on peut être amené à définir des systèmes d'évaluation utilisant des instants temporels auxiliaires de référence. On a proposé des modèles dans lesquels l'évaluation temporelle est faite par rapport à trois temps privilégiés : celui du locuteur, celui de l'événement décrit, et un temps de référence.

Ce formalisme permet de définir de nouveaux opérateurs qui s'appliquent à des paires ordonnées de points. L'opérateur "maintenant" ($[M]$) peut, par exemple, être évalué comme suit :

$$\mathcal{M} \models_{<t_1,t_2>} [M]A \text{ ssi } \mathcal{M} \models_{<t_1,t_1>} A.$$

De même, l'opérateur "étant", déjà rencontré, peut être défini comme un opérateur bi-dimensionnel :

$$\mathcal{M} \models_{<t_1,t_2>} \Pi A \text{ ssi, pour tout } t \text{ entre } t_1 \text{ et } t_2, \mathcal{M} \models_{<t,t>} A.$$

Les logiques temporelles à plusieurs dimensions peuvent servir à la construction des systèmes qui modélisent à la fois l'obligation et des données temporelles, ou des systèmes qui utilisent des données temporelles et historiques. Un de ces systèmes est brièvement examiné ci-dessous.

3.2.8.1. *Exemple de langage multimodal*

Un langage combinant les modalités temporelles (dans le *futur* et/ou dans le *passé*) et aléthiques (il est *nécessaire* que, il est *possible* que) est *multimodal*. Ses modalités atomiques sont les suivantes :

$[F]$: toujours dans le futur; $<F>$: parfois dans le futur;
$[P]$: toujours dans le passé; $<P>$: parfois dans le passé;
$\Box$: il est nécessaire que; $\Diamond$: il est possible que.

L'univers W est le produit cartésien de deux ensembles :

l'ensemble Ω des mondes possibles;
l'ensemble T des instants temporels.

$$W = M \times T = \{\omega_1, \cdots \omega_p\} \times \{t_1, \cdots t_n\}$$

Un élément de l'univers (ou *monde*) est une paire :

<monde possible, instant temporel>, soit :
$< \omega_i, t_j >$; $\omega_i \in \Omega$, $t_j \in T$.

A l'univers W sont associées des relations temporelles (vers le *passé* et/ou vers le *futur*) et une relation aléthique, soit :

$$R = (R_M, R_F, R_P).$$

En général on prendra pour R_F et R_P la relation d'ordre $<$, ce qui signifie que T est une chaîne :

tR_Ft' si et seulement si $t < t'$;
tR_Pt' si et seulement si $t' < t$.

Les modalités universelles sont alors définies de la façon suivante :

$\mathcal{M} \models_{<\omega,t>} [F]A$ ssi $\mathcal{M} \models_{<\omega,t'>} A$ $\forall t' : t < t'$;
$\mathcal{M} \models_{<\omega,t>} [P]A$ ssi $\mathcal{M} \models_{<\omega,t'>} A$ $\forall t' : t' < t$;
$\mathcal{M} \models_{<\omega,t>} \Box A$ ssi $\mathcal{M} \models_{<\omega',t>} A$ $\forall \omega' : \omega' R_M \omega$.

Donc, par définition de la nécessité, ce qui est nécessairement vrai à un instant donné et dans un monde donné est vrai à cet instant dans tous les mondes qui sont en relation avec le monde considéré.

3.2.9. *Logique d'intervalles*

Il peut être avantageux de représenter le temps au moyen d'intervalles plutôt que de points. Entre autres, les opérateurs qui formalisent les locutions "étant", "pendant" et "depuis", s'expriment naturellement au moyen d'intervalles plutôt que de points. A ces intervalles doivent être associées des *structures d'intervalles* auxquelles sont adjoints des opérateurs modaux appropriés. On peut, par exemple, décrire les relations entre intervalles au moyen des relations de précédence ($i < j$) et d'inclusion ($i \subseteq$). A cette dernière relation sur les intervalles peuvent être associés de nouveaux opérateurs modaux :

$\Box_{sous} A$: A est vérifié dans tous les sous-intervalles;
$\Box^{sur} A$: A est vérifié dans tous les sur-intervalles.

Ces opérateurs universels ont évidemment leur duals existentiels $\Diamond_{sous}$ et $\Diamond^{sur}$. Les propriétés de l'inclusion sur les intervalles peuvent alors se représenter en termes de ces nouvelles modalités :

Réflexivité : $\Box_{sous} A \supset A$;
Transitivité : $\Box_{sous} A \supset \Box_{sous} \Box_{sous} A$.

L'interaction entre l'inclusion sur les intervalles et la propriété de précédence temporelle s'exprime de la façon suivante :

$<F>A \supset \Box_{sous} <F>A$: monotonie à droite pour $<, \subseteq$;
$<P>A \supset \Box_{sous} <P>A$: monotonie à gauche pour $<, \subseteq$.

La logique de ces opérateurs est particulièrement bien adaptée pour représenter les notions de persistance [van Benthem 95]. Ces opérateurs permettent de traduire des locutions de la langue naturelle, telles "durant" et "pendant", qui expriment une certaine durée, et qui marquent donc aussi la persistance au cours d'un intervalle de temps. Par exemple, l'expression "étant occupé à travailler", serait rendue par $\Diamond^{sur} travailler$.

3.3. Logique épistémique

3.3.1. *Introduction*

L'épistémologie est la discipline qui a pour objet l'étude critique des sciences. L'épistémologie fait partie de la *théorie de la connaissance*; elle a son origine dans la Grèce antique, les philosophes s'y intéressant à des questions telles que : "que savons-nous?", "qu'est-ce qui est connaissable?", etc. La construction d'une logique formelle permettant de modéliser le raisonnement à propos des connaissances est récente, les premiers travaux datant des années cinquante.

Des domaines aussi divers que l'économie, la linguistique et l'intelligence artificielle utilisent le raisonnement sur les connaissances. On y rencontre des problèmes pratiques où l'action à effectuer dépend de la connaissance de l'état d'un certain domaine d'expertise. Par exemple : à quel moment un cambiste possède-t-il une connaissance suffisante des tendances du marché pour arrêter la récolte d'information et prendre une décision ?

La logique épistémique s'impose comme contrainte de modéliser des situations impliquant la connaissance d'un groupe d'agents, plutôt que d'un seul. Un agent ne doit pas seulement tenir compte des faits qui lui sont connus, mais également de la connaissance qu'en ont d'autres agents dans le groupe. Par exemple, un vendeur doit considérer que les acheteurs connaissent le prix normal d'une marchandise; tout acheteur doit aussi tenir compte de ce que le vendeur sait ce que l'acheteur sait à propos du prix de la marchandise. On se perd vite dans ce type de connaissance emboîtée. Or, c'est précisément ce genre de connaissance que l'on utilise lorsque l'on analyse la connaissance des agents dans un groupe.

3.3.2. *Les mondes possibles en logique épistémique*

Le modèle des mondes possibles en logique modale peut se transposer à l'interprétation épistémique. On souhaite ici représenter l'intuition que l'agent

i connaît la proposition A dans le monde w du modèle $\mathcal{M}$ dès que A est vraie dans tous les mondes que i considère comme possibles à partir de w. On voit donc qu'à la modalité $[K_i]$, signifiant "l'agent i sait que", on peut associer une relation d'accessibilité définissant quels mondes cet agent considère comme possibles. On dira donc que (s,t) appartient à la relation d'accessibilité associée à $[K_i]$ si l'agent i considère le monde t comme possible, étant donné les informations qu'il détient sur le monde s. Puisqu'on veut modéliser l'intuition que l'agent i considère t comme possible dans le monde s si l'agent i perçoit la même information s'il est dans s ou dans t, il s'agit d'une relation d'équivalence sur l'ensemble des mondes possibles. On dit que les mondes s et t sont *indiscernables pour l'agent i.* Ce qui ne veut pas dire que ces deux mondes soient identiques ! La structure ainsi définie peut être représentée au moyen d'un graphe étiqueté. Les nœuds du graphe sont les mondes possibles de l'univers; l'étiquette du nœud décrit les propositions de base qui sont vraies dans ce monde. L'arc reliant l'état s à l'état t est étiqueté i si (s,t) appartient à la relation associée à l'agent $[K_i]$. Un arc reliant deux états est donc étiqueté au moyen d'un ensemble d'agents.

On illustre cette définition à l'aide de l'exemple simple suivant (repris de [Fagin et al. 95], p. 19). Soit un modèle $\mathcal{M} = <W, R_1, R_2, V>$; W est un univers constitué de trois mondes notés s,t,u; R_1, R_2 sont les relations d'accessibilité respectivement associées aux agents 1 et 2; la fonction de valuation attribue à une proposition p la valeur "vraie" en s et u et la valeur "fausse" en t. De plus l'agent 1 ne peut distinguer s de t et l'agent 2 ne peut distinguer s de u. Cette situation est décrite par le graphe de la figure 3.2.

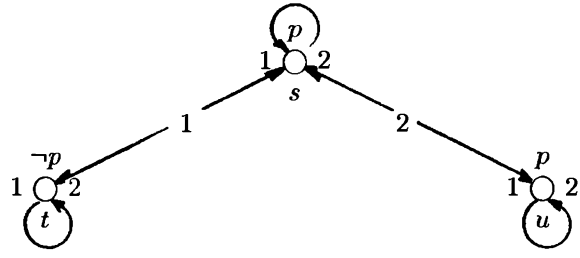


Figure 3.2 *Structure à deux agents*

Si p signifie "il y a du soleil", alors il y a du soleil dans l'état s mais l'agent 1 ne le sait pas, puisque dans cet état il considère à la fois s et t possibles, et ne peut les distinguer. Par contre, l'agent 2 dans l'état s sait qu'il y a du soleil, puisque dans les deux états qu'il considère comme possibles (s et u), il y a du soleil. Dans l'état t il sait également qu'il n'y a pas de soleil. Dès lors l'agent 1 sait que l'agent 2 sait ! Par contre, l'agent 2 ne sait pas que l'agent 1 ne sait pas qu'il y a du soleil (en effet, l'agent 2 en s ne peut pas distinguer s de u et en u l'agent 1 sait qu'il y a du soleil). Tout cet enchevêtrement de connaissances et d'ignorances se formalise comme suit :

$$\mathcal{M} \models_s (p \wedge \neg[K_1]p \wedge [K_2]p \wedge [K_1]([K_2]p \vee [K_2]\neg p) \wedge \neg[K_2]\neg[K_1]p).$$

Observons que la proposition p possède la même valeur de vérité, à la fois dans s et dans u. On ne peut toutefois éliminer un de ces états car un état est non seulement caractérisé par les propositions qui y sont vraies, mais également par les relations d'accessibilité sur les connaissances des agents. Or, dans notre exemple, l'agent s considère le monde t comme possible, alors que ce n'est pas le cas en u.

3.3.3. *Exemple : Le problème des trois conseillers*

L'objet de ce paragraphe est d'illustrer l'usage que l'on peut faire de la logique modale dans le contexte d'une formalisation des connaissances de différents agents et des raisonnements que ces agents peuvent tenir à propos de leurs connaissances. En particulier, les agents pourront se poser des questions à propos de leurs propres connaissances et de ce fait aboutir à des connaissances à propos de celles-ci. On parlera dans ce cas de "raisonnement introspectif". Nous allons illustrer la formalisation de ce type de raisonnement au moyen du problème classique des trois conseillers qui s'énonce comme suit.

Un roi désirant savoir lequel de ses trois conseillers est le plus sagace peint un point blanc sur le front de chacun d'eux. Le roi leur dit qu'il a peint un point blanc ou un point noir sur le front de chacun et qu'au moins un des points est blanc; il demande ensuite à chaque conseiller de deviner la couleur de son propre point. Après un temps de réflexion le premier répond qu'il ne sait pas; entendant cela le deuxième dit qu'il ne sait pas non plus. Après avoir entendu la réponse des deux premiers, le troisième déclare que son point est blanc.

Le problème des trois conseillers a été utilisé par [McCarthy 80, 86] pour tester diverses modélisations de la connaissance. Le type de raisonnement qu'il faut tenir dans ce problème est fort complexe; il faut notamment avoir un moyen de raisonner sur la connaissance des autres conseillers. Le raisonnement que tient le troisième conseiller est le suivant.

Admettons que mon point soit noir. Alors le deuxième d'entre nous devrait savoir que son point est blanc parce qu'il sait que s'il était noir alors le premier conseiller aurait vu deux points noirs et en aurait conclu que le sien était blanc. Comme aucun des deux premiers n'a pu deviner la couleur de son point, il faut que le mien soit blanc.

La difficulté du traitement logique de ce problème provient essentiellement du caractère imbriqué des raisonnements qu'il faut tenir. Le troisième conseiller doit raisonner sur ce que le deuxième ne sait pas (la couleur de son propre point); il doit également raisonner en tenant compte de ce que le deuxième conseiller raisonne sur ce que lui-même (le troisième conseiller) ne sait pas (la couleur du point du troisième conseiller). Il est particulièrement malaisé de se rappeler, au bout de quelques raisonnements imbriqués, qui sait qu'une tierce

personne ne savait pas qu'elle-même savait, etc. Il est fort difficile pour l'esprit humain de se retrouver dans un labyrinthe de croyances et de non-croyances imbriquées. C'est une des raisons pour lesquelles on a tenté d'automatiser les raisonnements au moyen de modèles logiques. Les logiques de croyance et de connaissance constituent une approche à cette question.

Le problème des trois conseillers a servi d'étalon à McCarthy pour tester diverses logiques de croyance et de connaissance. Konolige a utilisé une famille logique de croyances non introspectives pour modéliser une variante du problème des trois conseillers; dans cette variante, la réponse du troisième conseiller est la suivante : *Moi non plus je ne connais pas la couleur de mon point; mais si le troisième d'entre nous était plus sagace je la connaîtrais* [Konolige 86].

Nous allons formaliser le problème classique des trois conseillers au moyen d'une logique multimodale de connaissance. Dans cette logique l'opérateur $[S_i]$ signifie "le conseiller (ou le *sage*) i sait que"; elle devra posséder les schémas d'axiomes et les règles d'inférence que voici :

$[S_i](X \supset Y) \supset ([S_i]X \supset [S_i]Y)$	:	*distribution*
$[S_i]X \supset X$	:	*connaissance*
$\vdash X \Rightarrow \vdash [S_i]X$	:	*nécessité*
$\vdash X \supset Y$, et $\vdash X \Rightarrow \vdash Y$	:	*modus ponens*

L'axiome de distribution et les règles d'inférence de *modus ponens* et de *nécessité* nous permettent de conclure qu'un agent connaît toutes les conséquences logiques de ses connaissances. Nous pouvons exprimer cette propriété sous forme d'une nouvelle règle d'inférence; schématiquement :

$$(X \vdash Y), [S_i]X \vdash [S_i]Y \quad : \text{ omniscience logique}$$

Afin de rendre la formulation du raisonnement plus courte nous allons tout d'abord traiter une version simplifiée du problème où l'on suppose qu'il n'y a que deux conseillers; ces conseillers seront notés A et B. L'information dont nous disposons peut s'exprimer sous la forme des hypothèses suivantes.

(1) A et B savent que chacun peut voir le point se trouvant sur le front de l'autre, et donc :

(1a) Si A n'a pas de point blanc, B sait que A n'a pas de point blanc. Si $Blanc(x)$ est un prédicat qui prend la valeur *vrai* si x a un point blanc et la valeur *faux* autrement, l'hypothèse (1a) est formalisée comme suit : $\neg Blanc(A) \supset [S_B](\neg Blanc(A))$.

(1b) A sait (1a) : $[S_A](\neg Blanc(A) \supset [S_B](\neg Blanc(A))$.

(2) A et B savent chacun qu'au moins un d'entre eux a un point blanc et chacun d'eux sait que l'autre le sait, et donc :

(2a) A sait que B sait que si A n'a pas de point blanc, alors B en a un, soit : $[S_A]([S_B](\neg Blanc(A) \supset Blanc(B)))$.

(3) B déclare qu'il ne sait pas s'il a un point blanc et donc A sait que B ne sait pas quelle est la couleur de son point, soit : $[S_A](\neg[S_B](Blanc(B)))$.

Les hypothèses (1b), (2a) et (3) sont écrites aux trois premières lignes de la démonstration du théorème. Dans la liste ci-dessous, qui constitue la démonstration du théorème, chaque ligne est obtenue à partir de lignes précédentes et d'axiomes; les numéros de ces lignes ainsi que les axiomes utilisés dans la démonstration sont indiqués à droite.

1.	$[S_A](\neg Blanc(A) \supset [S_B](\neg Blanc(A)))$	
2.	$[S_A]([S_B](\neg Blanc(A) \supset Blanc(B)))$	
3.	$[S_A](\neg[S_B](Blanc(B)))$	
4.	$\neg Blanc(A) \supset [S_B](\neg Blanc(A))$	1, axiome de connaissance
5.	$[S_B](\neg Blanc(A) \supset Blanc(B))$	2, axiome de connaissance
6.	$[S_B](\neg Blanc(A)) \supset [S_B](Blanc(B))$	5, axiome de distribution
7.	$\neg Blanc(A) \supset [S_B](Blanc(B))$	4, 6, transitivité
8.	$\neg[S_B](Blanc(B)) \supset Blanc(A)$	reformulation de 7
9.	$[S_A](\neg[S_B](Blanc(B)) \supset Blanc(A))$	1, 2, 4, 5, 8, omniscience logique
10.	$[S_A](\neg[S_B](Blanc(B))) \supset [S_A](Blanc(A))$	9, axiome de distribution
11.	$[S_A](Blanc(A))$	3, 10, modus ponens

La ligne 9 de la démonstration peut se comprendre comme suit. Les hypothèses 1 et 2 signifient que A connaît les formules 4 et 5; des formules 4 et 5 on infère la formule 8; la règle d'omniscience logique nous permet donc d'inférer que A connaît la formule 8; on peut écrire cela schématiquement sous la forme :

$$(1.2 \vdash [S_A](4,5)), (4,5 \vdash 8) \vdash [S_A](8).$$

La formalisation des hypothèses 1–3 est celle proposée dans [Genesereth and Nilsson 87]. Dans la version des trois conseillers, il y a un niveau supplémentaire de connaissances imbriquées mais la stratégie de preuve est essentiellement la même (voir aussi [Konolige 86]).

3.3.4. *Connaissance commune et distribuée*

Des états nouveaux de la connaissance apparaissent naturellement dans une situation à plusieurs agents. On s'y intéresse souvent à des situations où chacun dans un groupe connaît un fait. Supposons que chacun connaisse la signification d'un feu rouge. Sachant cela, un conducteur est-il assuré de pouvoir rouler en toute sécurité ? La réponse est "non", à moins qu'il sache que tout le monde connaît cette règle et la suit. Même *l'état de connaissance* où chacun sait que chacun sait est insuffisant pour bon nombre d'applications. Dans certains cas, il est nécessaire de considérer l'état dans lequel, simultanément, chacun sait A, chacun sait que chacun sait A, chacun sait que chacun sait que chacun

sait A, etc. Dans ce cas, on dit que le groupe a une connaissance commune de A. Il s'agit d'une notion importante. Une connaissance commune est ce que "n'importe quel fou sait". Une connaissance commune s'avérant être un prérequis pour la réalisation d'un accord, cette notion joue un rôle-clé dans l'analyse de l'interaction d'agents dans un groupe. On dit également qu'un groupe a une *connaissance distribuée* d'un fait A si la connaissance de A est distribuée parmi les membres du groupe, de telle sorte que la mise en commun de la connaissance de tous ses membres permette au groupe de déduire A. La *connaissance distribuée* est celle d'un ***sage***, contrairement à la *connaisance commune*, qui est celle d'un *fou*.

Afin d'exprimer les notions de connaissance commune et connaissance distribuée, on étend le langage de la logique multimodale en y introduisant de nouveaux opérateurs modaux, $[\mathcal{E}_G]$ (chacun – *E*veryone – dans le groupe G sait que), $[C_G]$ (c'est une connaissance commune des agents du groupe G que), et $[\mathcal{D}_G]$ (c'est une connaissance distribuée des agents du groupe G que). Nous avons alors :

$$\mathcal{M} \models_w [\mathcal{E}_G]A \text{ ssi } \mathcal{M} \models_w [K_i]A \text{ pour chaque } i \in G.$$

La formule $\mathcal{M} \models_w [C_G]A$ est vraie si chacun dans G sait que A est vrai. si chacun dans G sait que chacun dans G sait que A est vrai, etc. Soit $[\mathcal{E}_G]^0 A = A$ et $[\mathcal{E}_G]^{k+1} A = [\mathcal{E}_G][\mathcal{E}_G]^k A$. Nous avons :

$$\mathcal{M} \models_w [C_G]A \text{ ssi } \mathcal{M} \models_w [\mathcal{E}_G]^k A \text{ pour } k = 1, 2, ...$$

Cette définition de la connaissance commune a une interprétation dans la théorie des graphes. On dit que l'état t est *G-accessible à partir de l'état s en k étapes* ($k \geq 1$) s'il existe une succession d'états $s_0, s_1, ..., s_k$ tels que $s_0 = s, s_k = t$, et que, pour tout j tel que $0 \leq j \leq k-1$, il existe $i \in G$, tel que l'arc $(s_j, s_{j+1}) \in [K_i]$. On dit que *t est G-accessible à partir de s en k étapes pour $k \geq 1$* s'il existe un chemin de s à t. Nous avons alors les propositions suivantes :

$\mathcal{M} \models_w [\mathcal{E}_G]^k A$ ssi $\mathcal{M} \models_t A$ pour tout t qui est G-accessible à partir de w en k étapes pour $k \geq 1$;
$\mathcal{M} \models_w [C_G]A$ ssi $\mathcal{M} \models_t A$ pour tout t qui est G-accessible à partir de w.

Un groupe G a une connaissance distribuée de A si la connaissance combinée des membres de G implique A. Nous avons :

$$\mathcal{M} \models_w [\mathcal{D}_G]A \text{ ssi } \mathcal{M} \models_t A \text{ pour tout } t \text{ tel que } (w, t) \in \bigcap_{i \in G} [K_i].$$

3.3.5. *Exemple : Le problème des trois conseillers (suite)*

Nous revisitons le problème des trois conseillers. Une forme élémentaire de ce problème avait été considérée au paragraphe 3.3.3. Nous en proposons ici une expression quelque peu plus élaborée qui s'appuie sur la définition d'opérateurs de connaissance commune et sur des temporalités.

Le langage L_w comprend trois symboles propositionnels, p_1, p_2, p_3; p_i signifie que le conseiller i a un point blanc. La résolution du problème s'effectue au cours d'une séquence de trois instants temporels : t_1, t_2, t_3.

Nous désignons de plus par $[S_0]$ la connaissance commune, et par $[S_1]$, $[S_2]$, $[S_3]$ celle des trois conseillers (voir [Konolige 86]).

1. A l'instant initial t_1, personne n'a encore parlé, sauf le roi qui a déclaré qu'au moins un des points est blanc. Les axiomes pour cet instant sont donc :

   (a) w_1 : $p_1 \wedge p_2 \wedge p_3$: chacun des trois conseillers a un point blanc.

   (b) w_2 : $[S_0, t_1](p_1 \vee p_2 \vee p_3)$: chacun sait au temps t_1 qu'au moins un des points est blanc.

   (c) w_3 : $(p_i \supset [S_j, t_1]p_i) \wedge [S_0, t_1](p_i \supset [S_j, t_1]p_i)$; si S_i a un point blanc, les autres le savent et chacun a la connaisance de cette connaissance.

   (d) w_4 : $(\neg p_i \supset [S_j, t_1]\neg p_i) \wedge [S_0, t_1](\neg p_i \supset [S_j, t_1]\neg p_i)$; si S_i n'a pas de point blanc, les autres le savent et chacun a la connaisance de cette connaissance.

   Les axiomes w_1–w_4 expriment des connaissances à propos de l'état initial du monde, et des connaissances à propos de ces connaissances; w_1 décrit le placement des points blancs; w_2 est le résultat de la déclaration du roi; w_3 et w_4 sont des schémas qui expriment que les sages sont capables d'évaluer les capacités de réflexion de chacun d'entre eux. Les axiomes suivants concernent des connaissances acquises respectivement aux instants t_2 et t_3.

2. A l'instant t_2 le deuxième sage parle et à l'instant t_3 le troisième. Les deux axiomes suivants L_1 et L_2 sont des axiomes d'apprentissage.

   (a) L_1 : $([S_1, t_1]p_1 \supset [S_0, t_2][S_1, t_1]p_1) \wedge (\neg[S_1, t_1]p_1 \supset [S_0, t_2]\neg[S_1, t_1]p_1)$. Ce que S_1 sait au temps t_1 à propos de la couleur de son point, les autres l'apprennent au temps t_2.

   (b) L_2 : $([S_2.t_2]p_2 \supset [S_0, t_3][S_2, t_2]p_2) \wedge (\neg[S_2, t_2]p_2 \supset [S_0, t_3]\neg[S_2, t_2]p_1)$. Après que le deuxième conseiller ait parlé, il est de la connaissance commune (au temps t_3) qu'il ne connaît pas la couleur de son point.

A ces axiomes on peut ajouter des axiomes qui décrivent la circonscription des connaissances des conseillers S_1 au temps t_1 et S_2 au temps t_2. Ces axiomes peuvent aisément être représentés au moyen de l'*opérateur de circonscription* introduit par Konolige. Cet opérateur s'écrit $<S_i : \Gamma>$, où Γ est un ensemble de formules; si p est une formule, l'expression $<S_i : \Gamma> p$ signifie que la formule p est dérivable par l'agent S_i à partir des formules Γ [Konolige 84]. L'opérateur de circonscription constitue un outil puissant pour représenter des situations

où la connaissance est bien délimitée. Par exemple, si l'on souhaite exprimer que "la seule connaissance que l'agent S_i possède à propos de la proposition p est celle qu'il peut déduire des faits Γ", nous écrirons :

$$<S_i : \Gamma> p \equiv [S_i]p.$$

Dire que l'agent S_i sait que p est vrai est équivalent à affirmer que l'agent S_i peut déduire p à partir de l'ensemble des faits Γ. Si l'implication directe $<S_i : \Gamma> p \supset [S_i]p$ est relativement évidente puisqu'elle exprime simplement que la connaissance de p est dérivable par l'agent S_i à partir de celle de Γ, soit : $[S_i]\Gamma \supset [S_i]p$, par contre l'implication inverse : $[S_i]p \supset <S_i : \Gamma> p$, est plus intéressante. Elle exprime que p ne peut être une connaissance de S_i, à moins que p ne soit dérivable de Γ. Ceci délimite bien l'information que l'agent S_i doit posséder pour pouvoir déduire p.

L'utilisation de l'opérateur de circonscription dans les logiques requiert encore l'introduction de la règle syntaxique suivante :

> Si p est une formule et Γ un ensemble fini de formules, alors $<S_i : \Gamma> p$ est une formule.

Revenons au problème des trois conseillers; nous pouvons écrire un premier axiome de circonscription :

$$C_1 : <S_i : w_2, w_3, w_4, p_j, p_k> p_i \equiv [S_i, t_1]p_i.$$

Cet axiome se réfère au système de connaissance de l'agent S_i. Dire que S_i connaît la couleur de son point au temps t_1 est équivalent de dire que S_i peut déduire la couleur de son point à partir des axiomes w_2, w_3, w_4 et de sa vision des points p_j et p_k. Nous utiliserons l'instance suivante de cet axiome qui décrit la circonscription des connaissances du conseiller S_1 au temps t_1 :

$$C_1 : <S_1 : w_2, w_3, w_4, p_2, p_3> p_1 \equiv [S_1, t_1]p_1.$$

Si S_1 peut déduire p_1 de w_2, w_3, w_4, et des deux points blancs qu'il voit, alors il connaît la couleur de son point; s'il ne peut pas déduire p_1, alors il ne la connaît pas.

On peut ensuite écrire un nouvel axiome qui décrit la circonscription des croyances du conseiller S_2 au temps t_2. La formulation de cet axiome est plus complexe puisque le conseiller S_2 a appris que le conseiller S_1, soit connaît, soit ne connaît pas la couleur de son point. Le nouvel axiome doit donc tenir compte de ces deux possibilités.

$$C_2 : \quad [S_1, t_1]p_1 \supset (<S_2 : w_2, w_3, w_4, [S_1, t_1]p_1, p_1, p_3> p_2 \equiv [S_2, t_2]p_2).$$

$$C_2 : \quad \neg[S_1, t_1]p_1 \supset (<S_2 : w_2, w_3, w_4, \neg[S_1, t_1]p_1, p_1, p_3> p_2 \equiv [S_2, t_2]p_2).$$

Ces axiomes, mis sous une forme légèrement modifiée, peuvent être convertis sous une forme clausale en vue de leur implémentation en Prolog.

w_1 : $p_1 \wedge p_2 \wedge p_3$
w_2 : $[S_0, t_1](p_1 \vee p_2 \vee p_3)$
w_3 : $[S_0, t_1](p_2 \supset [S_2, t_1]p_1)$
w_3 : $[S_0, t_1](p_2 \supset [S_3, t_1]p_1)$
w_3 : $[S_0, t_1](p_2 \supset [S_1, t_1]p_2)$
w_3 : $[S_0, t_1](p_2 \supset [S_3, t_1]p_2)$
w_3 : $[S_0, t_1](p_3 \supset [S_1, t_1]p_3)$
w_3 : $[S_0, t_1](p_3 \supset [S_2, t_1]p_3)$
w_3 : $[S_0, t_1](\neg p_1 \supset [S_2, t_1]\neg p_1)$
w_3 : $[S_0, t_1](\neg p_1 \supset [S_3, t_1]\neg p_1)$
w_3 : $[S_0, t_1](\neg p_2 \supset [S_1, t_1]\neg p_2)$
w_3 : $[S_0, t_1](\neg p_2 \supset [S_3, t_1]\neg p_2)$
w_3 : $[S_0, t_1](\neg p_3 \supset [S_1, t_1]\neg p_3)$
w_3 : $[S_0, t_1](\neg p_3 \supset [S_2, t_1]\neg p_3)$
w_4 : $p_1 \supset [S_2, t_1]p_1$
w_4 : $p_1 \supset [S_3, t_1]p_1$
w_4 : $p_2 \supset [S_1, t_1]p_2$
w_4 : $p_2 \supset [S_3, t_1]p_2$
w_4 : $p_3 \supset [S_1, t_1]p_3$
w_4 : $p_3 \supset [S_2, t_1]p_3$
w_4 : $\neg p_1 \supset [S_2, t_1]\neg p_1$
w_4 : $\neg p_1 \supset [S_3, t_1]\neg p_1$
w_4 : $\neg p_2 \supset [S_1, t_1]\neg p_2$
w_4 : $\neg p_2 \supset [S_3, t_1]\neg p_2$
w_4 : $\neg p_3 \supset [S_1, t_1]\neg p_3$
w_4 : $\neg p_3 \supset [S_2, t_1]\neg p_3$
L_1 : $[S_1, t_1]p_1 \supset [S_0, t_2][S_1, t_1]p_1$
L_1 : $\neg[S_1, t_1]p_1 \supset [S_0, t_2]\neg[S_1, t_1]p_1$
L_2 : $[S_2, t_2]p_2 \supset [S_0, t_3][S_2, t_2]p_2$
L_2 : $\neg[S_2, t_2]p_2 \supset [S_0, t_3]\neg[S_2, t_2]p_1$
PK_1 : $p_2 \wedge p_3$
C_1 : $<S_1 : w_2, w_3, w_4, PK_1> p_1 \equiv [S_1, t_1]p_1$
SK_1 : $[S_1, t_1]p_1$
SK_2 : $\neg[S_1, t_1]p_1$
PK_2 : $p_1 \wedge p_3$
C_2 : $[S_1, t_1]p_1 \supset (<S_2 : w_2, w_3, w_4, L_1, SK_1, PK_2> p_2 \equiv [S_2, t_2]p_2)$
C_2 : $\neg[S_1, t_1]p_1 \supset (< S_2 : w_2, w_3, w_4, L_1, SK_2, PK_2 > p_2 \equiv [S_2, t_2]p_2)$

Ces instructions sont ensuite initialisées au moyen de la clause but :

$\neg[S_3, t_3]p_3$.

Notons enfin qu'il peut être utile de considérer deux axiomes supplémentaires :

$w_5 : \neg[S_1, t_1]p_1 \wedge [S_0, t_2]\neg[S_1, t_1]p_1$.

$w_6 : \neg[S_2, t_2]p_2 \wedge [S_0, t_3]\neg[S_2, t_2]p_2$.

Ces axiomes étant par ailleurs dérivables de C_1, L_1, C_2, L_2, leur présence dans les hypothèses n'est pas indispensable dans le contexte de la résolution automatique du problème des trois conseillers.

3.3.6. *Connaissance et temporalités*

Comme l'exemple du paragraphe précédent nous le montre, les modalités de la connaissance sont souvent combinées avec celles du temps pour constituer des langages multimodaux pouvant représenter des connaissances qui évoluent avec le temps.

Nous nous situons alors dans une logique modale épistémique de type $S5$, dont les axiomes sont les suivants.

3.3.6.1. *Axiomes de la connaissance et règles d'inférence*

distributivité	:	$[K_i](A \supset B) \supset ([K_i]A \supset [K_i]B)$
connaissance	:	$[K_i]A \supset A$
introspection positive	:	$[K_i]A \supset [K_i][K_i]A$
introspection négative	:	$\neg[K_i]A \supset [K_i]\neg[K_i]A$
nécessité	:	si $\vdash A$, alors $\vdash [K_i]A$
modus ponens	:	si $\vdash A$ et $\vdash A \supset B$ alors $\vdash B$

Ce système contient les théorèmes suivants qui énoncent des propriétés des opérateurs de connaissance commune :

1. $[\mathcal{E}_G]A \equiv \wedge_{i \in G}[K_i]A$;
2. $[\mathcal{C}_G]A \supset [\mathcal{E}_G](A \wedge [\mathcal{C}_G]A)$.

Le système axiomatique d'un langage multimodal épistémique et temporel contient aussi des axiomes et règles d'inférence pour caractériser les propriétés du temps.

Par exemple, dans le cas de la logique temporelle linéaire :

distributivité	:	$\bigcirc(A \supset B) \supset (\bigcirc A \supset \bigcirc B)$
déterminisme	:	$\bigcirc\neg A \equiv \neg\bigcirc A$
définition de until	:	$A \mathcal{U} B \equiv B \vee (A \wedge \bigcirc(A \mathcal{U} B))$
nécessité	:	si $\vdash A$ alors $\vdash \bigcirc A$

Dans les systèmes où on exige en plus des agents d'avoir une mémoire parfaite, il faudra ajouter l'axiome :

mémoire parfaite : $[K_i]\Box A \supset \Box[K_i]A$.

Cet axiome indique que si un agent K_i sait que A est toujours vrai, alors il est toujours vrai que l'agent K_i sait que A est vrai.

La propriété de synchronisme, c'est-à-dire le fait que le temps s'écoule de la même façon pour tous les agents, est modélisée par l'axiome :

synchronisme : $[K_i]\bigcirc A \supset \bigcirc[K_i]A$.

Situons-nous maintenant dans un système synchrone où tous les agents ont une mémoire parfaite; si on ajoute aussi la propriété selon laquelle l'assignation des valeurs de vérité pour les atomes propositionnels ne change pas au cours du temps, il faut introduire l'axiome suivant :

persistance de l'information : $\Box p \vee \Box\neg p, \forall p.$

Intéressons-nous enfin à la *connaissance distribuée* de formules propositionnelles (formules qui ne contiennent pas d'opérateurs modaux). Cette connaissance ne peut ni croître, ni décroître dans le groupe de tous les agents; formellement, si A est une formule propositionnelle, on a :

persistance positive : $[\mathcal{D}_G]A \supset \Box[\mathcal{D}_G]A$;

persistance négative : $\neg[\mathcal{D}_G]A \supset \Box\neg[\mathcal{D}_G]A$.

Ces deux axiomes entrent dans la catégorie des *axiomes relatifs aux agents observateurs.*

Chapitre 4
Logique modale quantifiée

4.1. Introduction

La logique modale quantifiée est considérée comme difficile et complexe. Les principes de la logique quantifiée ne peuvent pas simplement être ajoutés à ceux de la logique modale pour former la logique modale quantifiée. La sémantique des formules contenant des quantificateurs dépend de différents choix préalables. Il s'agit tout d'abord de choisir les types d'éléments sur lesquels porte la quantification; on peut, comme en logique prédicative classique, faire porter la quantification sur des variables individuelles; on peut également décider de la faire porter sur l'*intension* de ces variables, ou même sur des types intensionnels plus complexes. Il faut ensuite décider de la sémantique des termes utilisés dans les expressions formelles; les termes pourront, par exemple, toujours dénoter la même entité dans tous les mondes de l'univers : ce sont les termes *rigides*; des termes pourront également dénoter des entités différentes selon le monde dans lequel on se situe : ce sont les termes *non rigides*. Il faut enfin choisir les caractéristiques des domaines de quantification; ces domaines peuvent soit être dépendants des mondes (domaines variables), soit en être indépendants (domaines fixes). Ces trois types de choix pouvant être effectués de façon indépendante les uns des autres, le nombre de logiques modales présentant de l'intérêt est très élevé.

Le but de ce chapitre est de présenter un aperçu des diverses sémantiques qui correspondent aux diverses options dont nous venons de faire mention.

4.2. Syntaxe et sémantique

4.2.1. *Syntaxe*

La syntaxe de la logique modale quantifiée est basée sur celle de la logique des prédicats (du premier ordre), dont les formules de base, ou *atomes*, sont les *formes prédicatives*. On obtient les formules de la logique modale quantifiée à partir des formes prédicatives et des règles d'induction qui font usage des connecteurs logiques $\neg, \wedge, \vee, \supset, \equiv$, des quantificateurs $\forall, \exists$, et des modalités $\Box, \Diamond$:

- **Base** : Tout atome (forme prédicative) est une formule.
- **Induction** : Si A et B sont des formules et si x est un symbole de variable, alors :
  - $\neg A$, $A \wedge B$, $A \vee B$, $A \supset B$, $A \equiv B$;
  - $\forall x A$, $\exists x A$;
  - $\Box A$ et $\Diamond A$;

  sont des formules.
- **Fermeture** : Une formule s'obtient uniquement à partir des règles de base et d'induction.

Dans la suite, on supposera que les logiques possèdent le symbole de prédicat binaire d'égalité noté $=$. Si t_1 et t_2 représentent des termes, alors on notera parfois $(t_1 = t_2)$ sous la forme prédicative préfixée $=(t_1, t_2)$.

4.2.2. *Sémantique*

La sémantique de la logique modale quantifiée se base sur celles de la logique modale propositionnelle et de la logique prédicative (premier ordre). Dans la théorie des modèles, ces sémantiques sont définies via les notions de *structure* et de *monde possible* (qui gèrent la composante modale de la logique) et de *fonction de valuation* (qui gère soit la composante propositionnelle, soit la composante prédicative de la logique).

La structure en logique modale quantifiée est identique à celle définie dans le cadre de la logique modale propositionnelle.

Par contre, la fonction de valuation n'est plus identique à celle définie dans le cadre de la logique des prédicats. Dans un contexte purement prédicatif, cette fonction, dont l'objet est l'attribution d'une valeur sémantique aux diverses expressions de la logique, associe aux constantes individuelles et aux variables individuelles des valeurs prises dans un ensemble d'individus appelé le *domaine du discours*; cet ensemble est noté S. Dans un contexte modal, cette fonction assigne aux diverses expressions de la logique, des valeurs (sémantiques) qui dépendent des mondes possibles. La fonction de valuation, notée V, devient,

dans un contexte modal, une fonction à deux arguments, le premier prenant ses valeurs dans le domaine prédicatif, le second prenant ses valeurs dans l'univers W. Ceci permet de concevoir formellement un *modèle* $\mathcal{M}$ pour la logique modale quantifiée comme un quadruplet $<W, R, S, V>$.

La fonction V associe :

- à une constante ou à une variable individuelle a une fonction de W dans S, notée $V(a)(w)$;
- à une constante fonctionnelle n-aire f une fonction de W dans $S^n \to S$, notée $V(f)(w)$;
- à une constante prédicative n-aire P une fonction de W dans $S^n \to \{\mathbf{V}, \mathbf{F}\}$, notée $V(P)(w)$.

La fonction V est étendue à d'autres constructions du langage au moyen des règles suivantes.

La fonction V associe :

- à une forme fonctionnelle $f(t_1, \cdots, t_n)$ la fonction F de W dans S : $F(w) =_{def} V(f)(w)(V(t_1)(w), \cdots, V(t_n)(w))$;
- à une forme prédicative $P(t_1, \cdots, t_n)$, la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$: $F(w) =_{def} V(P)(w)(V(t_1)(w), \cdots, V(t_n)(w))$: $F(w)$ est $\mathbf{V}$ si et seulement si $\{V(t_1)(w), V(t_2)(w), \cdots, V(t_n)(w)\} \in V(P)(w)$ et $F(w)$ est $\mathbf{F}$ dans tous les autres cas;
- à une formule de la forme $\neg A$ la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$: $F(w)$ est $\mathbf{V}$ si et seulement si $V(A)(w)$ est $\mathbf{F}$;
- à une formule de la forme $A \vee B$ la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$: $F(w)$ est $\mathbf{V}$ si et seulement si $V(A)(w)$ est $\mathbf{V}$ ou $V(B)(w)$ est $\mathbf{V}$;
- à une formule de la forme $A \wedge B$ la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$: $F(w)$ est $\mathbf{V}$ si et seulement si $V(A)(w)$ est $\mathbf{V}$ et $V(B)(w)$ est $\mathbf{V}$;
- à une formule de la forme $A \supset B$ la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$: $F(w)$ est $\mathbf{V}$ si et seulement si $V(A)(w)$ est $\mathbf{F}$ ou $V(B)(w)$ est $\mathbf{V}$;
- à une formule de la forme $A \equiv B$ la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$: $F(w)$ est $\mathbf{V}$ si et seulement si $V(A)(w) = V(B)(w)$;
- à une formule de la forme $\Box A$ la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$: $F(w)$ est $\mathbf{V}$ si et seulement si $V(A)(w')$ est $\mathbf{V}$ pour tout $w' \in W$ tel que wRw';
- à une formule de la forme $\Diamond A$ la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$: $F(w)$ est $\mathbf{V}$ si et seulement si $V(A)(w')$ est $\mathbf{V}$ pour au moins un $w' \in W$ tel que wRw'.

Pour que la fonction V soit étendue à toutes les constructions du langage, il reste à établir les règles pour les formules $\forall x A$ et $\exists x A$.

Ce sont les différents choix qui peuvent être faits concernant :

- les éléments sur lesquels on fait porter la quantification (le type des variables x dans les formules $\forall x X$ et $\exists x X$);

- la sémantique des termes (termes rigides ou termes non rigides);
- le domaine de quantification (domaine fixe ou domaines dépendant des mondes);

qui introduisent la complexité dans la logique modale quantifiée. Entre autres choses, il faut noter que le domaine de quantification, noté Q, ne doit plus nécessairement correspondre au domaine du discours comme c'était le cas en logique des prédicats. Le domaine Q peut ainsi être différent du domaine S, et le modèle $\mathcal{M}$ comportera donc un élément supplémentaire; en toute généralité le modèle sera un quintuplet constitué comme suit :

$$\overbrace{\mathcal{M}}^{\textit{composantes}} = < \overbrace{W,R}^{\textit{modale}}, \overbrace{S}^{\textit{prédicative}}, \overbrace{Q,V}^{\textit{modale}} >.$$

Ce modèle contient une composante proprement modale comprenant l'univers W, la relation d'accessibilité R, le domaine de quantification Q et la fonction de valuation V (supposée inclure la fonction d'assignation), et une composante proprement prédicative, le domaine du discours S.

4.2.3. *Extension et intension*

La plupart des expressions d'une langue naturelle comme le français ou l'anglais, comportent deux types de valeurs sémantiques, auxquelles on donne respectivement les noms de *sens* et de *référence*. La sémantique de la logique modale quantifiée, telle que nous l'avons définie, rend compte de ces deux types de valeurs sémantiques propres à la langue naturelle. On peut illustrer la différence existant entre "sens" et "référence" au moyen de l'exemple suivant. Considérons la phrase : "Jean travaille". Que peut-on dire concernant sa valeur sémantique ?

1. Dans certains cas on s'intéresse à la *valeur de vérité* de la phrase. Cette phrase est-elle vraie ou fausse dans un modèle donné (dans un contexte donné, dans un monde donné) ? On considère qu'elle est vraie si l'individu dénoté par "Jean" appartient à la classe des individus qui travaillent (dans un monde donné à un moment donné).
2. Dans d'autres cas on s'intéresse plutôt au *sens* de la phrase. Que veut dire cette phrase ? On est souvent incapable de déterminer sa valeur de vérité (Jean travaille parfois, mais pas toujours), mais on est normalement capable de comprendre son sens, c'est-à-dire la proposition qu'elle véhicule.

Nous dirons donc que le *sens* d'une phrase est la proposition, la pensée qu'elle véhicule, tandis que sa *référence* est sa valeur de vérité.

Cette valeur de vérité est fonction de la réalité (du modèle) à laquelle cette phrase se réfère. Généralement nous ne connaissons pas la référence, la dénotation, des composants de la phrase, mais seulement leur sens, leur compréhen-

sion. C'est pourquoi nous pouvons comprendre une phrase sans savoir si elle est vraie.

On peut étendre cette distinction à pratiquement tous types d'expressions de la langue naturelle. Pour un nom commun, tel que "président" par exemple, son sens est l'ensemble des attributs attachés à la fonction de président; sa référence est l'ensemble des personnes qui occupent cette fonction.

"Sens" et "référence" constituent donc deux concepts sémantiques de la langue naturelle dont il importe de définir les équivalents en langue formelle. Dans le contexte d'une formalisation, on utilise les termes *intension* et *extension* respectivement à la place de *sens* et de *référence.*

Tout d'abord, l'*extension d'une expression dans un monde possible* est, par définition, la valeur sémantique de cette expression dans ce monde. C'est Carnap qui le premier a tenté de formaliser la notion de sens [Carnap 36]. Il suggère de définir l'intension d'une expression comme l'ensemble de toutes les extensions que cette expression peut avoir (dans l'univers des mondes possibles). Chez Montague, cet ensemble sera organisé comme une fonction dont le domaine est l'ensemble de tous les mondes possibles de l'univers et dont le codomaine est un ensemble d'extensions [Montague 74a-d] (voir aussi [Gochet 82a-b], [Hendriks 89], [Janssen 86], [Muskens 88], [Thayse et al. 89], [Thayse et al. 91]) .

La définition suivante exprime de façon formelle, au moyen de la fonction de valuation V, les concepts d'intension et d'extension pour tous les types d'expressions de la logique modale quantifiée pour lesquels la fonction V est définie.

Définition 4.1. L'*intension* d'une expression C du langage est le résultat de l'application de la fonction V à C et l'*extension* de C dans le monde $w \in W$ est le résultat de l'évaluation de l'intension de C en w.

4.2.4. *Désignateurs rigides et non rigides*

Dans certaines sémantiques, les *termes* sont des *désignateurs rigides*, ce qui signifie que leurs extensions (valeurs sémantiques) sont les mêmes dans tous les mondes possibles. Soit, par exemple, les constantes individuelles c_1, c_2, c_3 représentant respectivement les individus Giscard, Mitterrand et Chirac :

$V(c_1)(w)$ = Giscard (pour tout w);
$V(c_2)(w)$ = Mitterrand (pour tout w);
$V(c_3)(w)$ = Chirac (pour tout w).

La personne nommée Giscard reste Giscard dans tous les mondes. Les désignateurs rigides sont caractérisés par des intensions constantes.

Plus généralement, les extensions des termes peuvent varier selon les mondes; dans ce cas, les termes sont des *désignateurs non rigides.* Soit par exemple le

terme c_4 qui représente "le président" :

$V(c_4)(w_1)$ = Giscard;
$V(c_4)(w_2)$ = Mitterrand;
$V(c_4)(w_3)$ = Chirac.

Toutes les extensions de c_4 ne sont pas identiques. L'ensemble des extensions constitue l'intension du terme. Les intensions sont des fonctions de l'ensemble des mondes possibles vers les extensions dans ces mondes.

Dans l'exemple, nous aurons donc :

$$\begin{aligned} V(c_4) \quad = \quad & \{w_1 \rightarrow V(c_4)(w_1) = \text{Giscard}, \\ & w_2 \rightarrow V(c_4)(w_2) = \text{Mitterrand}, \\ & w_3 \rightarrow V(c_4)(w_3) = \text{Chirac}\}; \end{aligned}$$

$$\begin{aligned} V(c_1) \quad = \quad & \{w_1 \rightarrow V(c_1)(w_1) = \text{Giscard}, \\ & w_2 \rightarrow V(c_1)(w_2) = \text{Giscard}, \\ & w_3 \rightarrow V(c_1)(w_3) = \text{Giscard}\}. \end{aligned}$$

Le terme c_1 est un désignateur rigide tandis que le terme c_4 est un désignateur non rigide.

4.2.5. *Les choix en logique modale quantifiée*

Les notions d'extension et d'intension, de désignateurs rigide et non rigide, vont nous permettre de préciser formellement les choix sémantiques qu'il faut opérer en vue de définir la sémantique des formules quantifiées de la logique modale prédicative.

- Choix 1 : *Sur quels types d'éléments porte la quantification ?*
  - La quantification se fait sur des *objets*; dans ce cas les variables sur lesquelles porte la quantification ont des intensions constantes. Formellement, le type de ces variables est e (pour "entity"); ce sont donc les *variables individuelles* de la logique du premier ordre. Il s'agit de la quantification la plus simple et la plus naturelle.
  - La quantification se fait sur des *concepts*; dans ce cas les variables sur lesquelles porte la quantification reçoivent des intensions non constantes; ce sont des fonctions des mondes possibles vers des objets. Ce type de quantification est introduit pour résoudre certaines difficultés qui apparaissent lorsque l'on quantifie sur des objets. En particulier, quantifier sur des concepts plutôt que sur des objets permet de maintenir une adéquation entre la nature non rigide des termes et le type des variables logiques.
  - La quantification se fait sur des *substances*; dans ce cas on introduit un *ensemble d'intensions de termes* pour chaque monde w, ensemble qui servira de domaine de quantification pour ce monde et que l'on

identifiera à l'ensemble des *substances* qui y sont définies. A nouveau, ce type de quantification est introduit dans le but de surmonter certaines incohérences qui apparaissent lorsque l'on quantifie sur des objets ou sur des concepts. Ces incohérences concernent essentiellement des interprétations inadéquates d'expressions formelles.

- Choix 2 : *Quelle est la sémantique des termes ?*
  Après avoir choisi le type des expressions sur lesquelles porte la quantification, on peut encore faire le choix sémantique suivant :

  - Tous les termes sont des *désignateurs rigides*; il s'agit d'un choix simple, mais aussi malheureusement simpliste, et qui peut ne pas correspondre à la réalité que l'on cherche à formaliser (en l'occurrence la langue naturelle), les termes des langues naturelles étant en général des désignateurs non rigides.
  - Les termes sont des *désignateurs non rigides*; ce choix cadre mieux avec la sémantique des langues naturelles. L'introduction de termes non rigides dans une logique formelle complique cependant l'axiomatisation de la langue.

- Choix 3 : *Quelle est la caractéristique du domaine de quantification ?*

  - Le domaine de quantification peut être *fixe* (indépendant du monde); cela signifie que les mêmes éléments du domaine existent dans tous les mondes possibles. Ceci ne correspond en général pas à la réalité que l'on souhaite modéliser. Si, par exemple, les mondes possibles représentent des moments temporels, il est clair que les individus ne sont pas éternels, et que donc différents individus doivent pouvoir exister à différentes époques.
  - Le domaine de quantification est constitué d'un ensemble de domaines dont chacun est associé à un monde (domaines *variables* ou *dépendant du monde*). Ceci correspond mieux à la réalité que l'on souhaite représenter, mais complique la sémantique de la langue formelle.

La figure 4.1 représente schématiquement ces différents choix.

Les logiques modales quantifiées dont il sera question ci-après constituent toutes des extensions prédicatives des logiques modales propositionnelles dont les systèmes axiomatiques sont formés d'un sous–ensemble des schémas d'axiomes K, B, T, 4, 5. Ces logiques modales propositionnelles sont toutes adéquates et complètes par rapport aux structures possédant les propriétés correspondantes (voir le chapitre 2).

Les deux sections suivantes présentent une vue des diverses sémantiques qui correspondent aux options qui viennent d'être évoquées. (Le contenu de ces

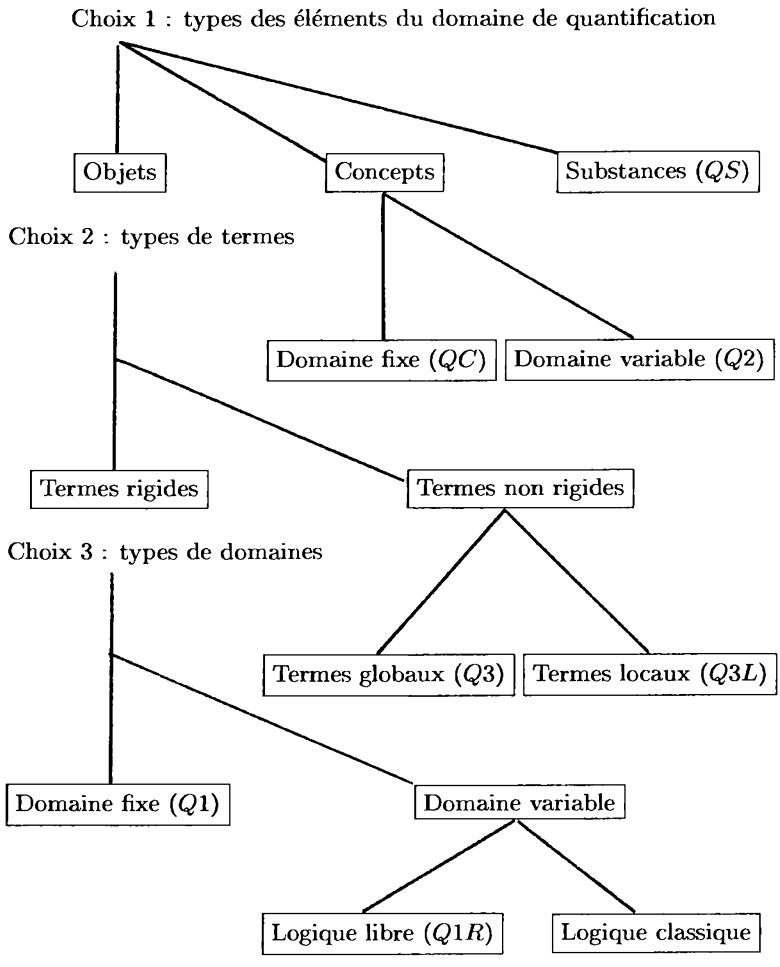


Figure 4.1 *Logiques modales quantifiées*

sections est inspiré de l'article de Garson dans l'encyclopédie de [Gabbay and Guenthner 84].)

4.3. Interprétation objectuelle

Un des choix les plus significatifs que l'on peut faire à propos des logiques modales quantifiées est celui qui concerne le *type* des éléments du domaine de quantification. Le cas le plus simple et le plus intuitif est celui où ces éléments sont des *objets*. Ce cas correspond à la logique des prédicats du premier ordre où le type des variables sur lesquelles porte la quantification est e ("entity").

4.3.1. *Termes rigides; domaines fixes*

La sémantique la plus simple est obtenue en imposant à l'intension de chaque terme d'être une fonction constante. Tous les termes sont par conséquent des *désignateurs rigides.* Cette condition de rigidité des termes – condition notée (*VRT*) – s'écrit :

(*VRT*) : $V(t)(w)$ est $V(t)(w')$ pour tout $w, w' \in W$.

La condition de rigidité traduit le fait que les noms propres – formalisés par les termes élémentaires que sont les constantes individuelles – ont des intensions constantes.

Imposer aux termes d'avoir des intensions constantes permet cependant encore aux constantes prédicatives d'avoir des intensions non constantes. La condition de rigidité sur les termes valide les deux règles suivantes – règles notées *RT* (pour *rigid terms*) – :

$$(RT) : \frac{t = t'}{\Box(t = t')}; \qquad \frac{\neg(t = t')}{\Box(\neg(t = t'))}.$$

Ces deux règles garantissent la substitution des termes identiques dans toutes les formules (même dans les expressions intensionnelles utilisées par exemple dans un contexte de discours indirect).

La condition de rigidité peut également être formalisée sous la forme de schémas d'axiomes :

$(t = t') \supset \Box(t = t')$;
$\neg(t = t') \supset \Box(\neg(t = t'))$.

Le choix suivant concerne le domaine de quantification. Dans l'approche dite *domaine fixe* – envisagée dans ce paragraphe – on suppose que le domaine de quantification est fixe et contient tous les objets auxquels on peut faire référence. L'approche dite *domaine variable* suppose, elle, que pour chaque monde, le domaine de quantification contient uniquement les objets qui existent dans ce monde; dans cette option, le contenu du domaine de quantification est une fonction des mondes; cette option sera envisagée ultérieurement.

4.3.1.1. *Le système Q1*

Bien qu'un système avec termes rigides et domaine fixe ne possède pas le degré de généralité requis pour représenter valablement les situations se présentant dans le monde réel, il est cependant intéressant de par la simplicité de sa sémantique. En effet, un tel système est constitué des règles sémantiques de la logique modale de base auxquelles il faut adjoindre celles relatives à la quantification en logique prédicative. Un *modèle objectuel avec domaine fixe*

et termes rigides (ou modèle $Q1$) est un quintuplet $<W, R, S, Q1, V>$ où W est l'univers des mondes possibles, R la relation d'accessibilité, S le domaine du discours, $Q1$ le domaine de quantification, avec $Q1 = S$, et V la fonction de valuation. Cette fonction satisfait à la condition (*VRT*) assurant que les intensions des termes sont constantes; elle est définie comme suit pour les formules quantifiées :

> **Règle** *Q1* :
> La fonction V associe à une formule de la forme $\forall x A$ la fonction F de W dans {**V**, **F**} telle que $F(w)$ est **V** si et seulement si $V_{x/d}(A)(w)$ est **V** pour tout $d \in S$,

la fonction $V_{x/d}$ étant la fonction identique à V sauf que l'intension donnée à x par $V_{x/d}$ est la fonction constante égale à d ($V_{x/d}$ est appelée *x-variante de* V).

Il est ensuite aisé de définir un système formel qui est adéquat pour les règles de la sémantique. Un tel système est décrit sous forme d'un énoncé de théorème :

Théorème 4.1. Soit $\mathcal{L}$ une logique modale propositionnelle qui est adéquate et complète par rapport à un ensemble de structures de Kripke. On obtient une logique modale quantifiée, notée $Q1\mathcal{L}$, qui est adéquate par rapport à la sémantique $Q1$ en ajoutant à $\mathcal{L}$ les ingrédients suivants :

- les axiomes *PL* de la logique des prédicats sans égalité;

- les règles *ID* de substitution des identiques :

$$t = t, \qquad \frac{t = t'}{P(t) \supset P(t')};$$

- les règles (*RT*) des termes rigides :

$$\frac{t = t'}{\Box(t = t')}, \qquad \frac{\neg(t = t')}{\Box(\neg(t = t'))};$$

- la *formule de Barcan* :

$$\forall x(\Box A) \supset \Box(\forall x A).$$

Remarque. L'introduction de la formule de Barcan dans un système axiomatique revient à imposer une contrainte entre la relation d'accessibilité R et un prédicat, noté E, où $E(x)(w)$ signifie "x existe dans le monde w" :

$$\forall w v(wRv \supset \forall x(E(x)(v) \supset E(x)(w))).$$

Ce qui signifie que si le monde w est en relation avec le monde v, alors si x existe dans v, x existera aussi dans w.

Certaines logiques contiennent ce symbole de prédicat unaire E dont le domaine sémantique (ou ensemble des extensions possibles) représente le *domaine des existants*. Les logiques qui contiennent ce prédicat unaire E sont

appelées *logiques libres*. Pour ces logiques, l'ensemble caractérisé par $V(E)$ peut être pris comme *domaine de quantification*.

Si $V(E)$ est une fonction constante, le domaine de quantification est le même dans chaque monde possible (domaine de quantification fixe). Dans ce cas, l'intension de E est une constante. Si $V(E)$ est une fonction des mondes possibles, le domaine de quantification dépend du monde dans lequel on se situe (domaine de quantification variable).

4.3.2. *Termes rigides; domaines dépendant du monde*

Le choix du "domaine fixe" n'est pas satisfaisant dans le contexte des expressions quantifiées de la langue naturelle. Quand on dit : "il existe un homme qui a signé la déclaration d'indépendance des Etats-Unis", on ne peut pas interpréter "il existe" au temps où la phrase est énoncée. De même, si un professeur annonce au cours : "tout le monde a réussi", cela concerne uniquement les étudiants qui suivent son cours, et non pas vraiment "tout le monde". Donc la "quantification au temps présent" devrait être dépendante du monde, ce qui reflète simplement le fait que les êtres ne sont pas immuables.

On peut démontrer que, dans le cadre du domaine fixe, $\forall x \Box \exists y(y = x)$ est une formule valide. Cette formule est la traduction de : "chaque chose existe éternellement". Or on souhaite en général que différentes choses puissent exister dans les différents mondes possibles. Cet objectif peut être atteint en construisant *un* monde par contexte et en attribuant des domaines de quantification différents pour chacun de ces mondes (*domaines variables*).

4.3.2.1. *Le système* $Q1R$

Un modèle orienté objet avec termes rigides et domaines dépendant du monde (ou modèle $Q1R$) est un quintuplet $< W, R, S, Q1R, V >$ où $Q1R$ est une fonction qui assigne un sous-ensemble $S(w)$ de S à chaque monde possible w, et où V satisfait la condition (*VRT*). La sémantique des formules quantifiées universellement est définie comme suit :

> **Règle** $Q1R$:
> La fonction V associe à une formule de la forme $\forall x A$ la fonction F de W dans {**V**, **F**} telle que $F(w)$ est **V** si et seulement si $V_{x/d}(A)(w)$ est **V** pour tout $d \in S(w)$.

Si la logique est *libre*, le langage possède le symbole de prédicat unaire E tel que la formule $E(t)$ est interprétée comme vraie dans w si et seulement si l'extension de t dans w appartient à $S(w)$:

$$V(E(t)) \text{ est } \mathbf{V} \text{ si et seulement si } V(t)(w) \in S(w).$$

On peut définir une logique adéquate (notée $Q1R$) qui valide la règle $Q1R$ en ajoutant aux axiomes et règles d'inférence de la logique libre minimale (notée

MFL), les axiomes et les règles d'inférence de la logique modale considérée, ainsi que les règles *ID* de substitution des identiques et *RT* relatives aux termes rigides. Cette logique est décrite sous forme de l'énoncé du théorème 4.2.

Théorème 4.2. Soit $\mathcal{L}$ une logique modale propositionnelle qui est adéquate et complète par rapport à un ensemble de structures de Kripke. On obtient une logique modale quantifiée *Q1R* en ajoutant à $\mathcal{L}$ les ingrédients suivants ($P(x)$ est une formule et $P(t)$ le résultat de la substitution des occurrences de x dans $P(x)$ par t) :

- la règle d'instantiation universelle de la logique libre :

$$(FUI) \ : \frac{\forall x P(x)}{E(t) \supset P(t)}$$

pour tout terme t;

- la règle de généralisation de la logique libre :

$$(FUG) \ : \frac{A \supset (E(t) \supset P(t))}{A \supset \forall x P(x)}$$

à condition que t n'apparaisse pas dans $A \supset \forall x P(x)$;

- les règles *ID* de substitution des identiques :

$$t = t. \qquad \frac{t = t'}{P(t) \supset P(t')};$$

- les règles (*RT*) des termes rigides :

$$\frac{t = t'}{\Box(t = t')}, \qquad \frac{\neg(t = t')}{\Box(\neg(t = t'))}.$$

4.3.2.2. *Remarques*

1. A propos de la logique libre

En logique classique, à partir de l'axiome de généralisation existentielle :

$P(t) \supset \exists x P(x)$,

et de l'identité $(a = a)$, on déduit :

$(a = a) \supset \exists x(x = a)$.

En logique classique on peut donc prouver l'existence de n'importe quoi : il suffit de pouvoir nommer quelque chose pour qu'automatiquement cette chose existe.

Ce problème résulte de ce que, dans la sémantique de la quantification, on suppose que toute constante a se réfère à un objet dans le domaine de quantification. La logique libre évite la difficulté en omettant l'hypothèse que

chaque nom (terme) doit se référer à un objet du domaine de quantification. En logique libre, l'axiome de généralisation existentielle (*free existential generalisation*) s'écrit :

$$(FEG) : (P(t) \wedge E(t)) \supset \exists x P(x).$$

De cet axiome et de $(a = a)$ on déduit :

$$(a = a) \wedge E(a) \supset \exists x(x = a),$$

et l'existence de a est assurée pour autant que la formule $E(a)$ soit vérifée, c'est-à-dire pour autant que a existe.

Notons encore que nous aurions pu définir le prédicat E par la relation :

$$E(t) \Longleftrightarrow \exists x(x = t).$$

2. Problèmes liés à la logique classique dans un contexte de domaines variables

Il est possible, dans le contexte d'une sémantique utilisant des domaines dépendant du monde, de conserver un système axiomatique d'une logique classique (et donc d'abandonner le cadre de la logique libre). Cependant, de sérieux problèmes apparaissent si l'on veut imposer au système formel d'être adéquat et complet.

Un premier problème résulte du fait que $\exists x(x = t)$ est un théorème de la logique des prédicats. Si le système formel modal était adéquat, il faudrait donc que $\exists x(x = t)$ soit une formule valide, ce qui n'est possible que si l'extension de t dans un monde w appartient à $S(w)$, quel que soit le monde $w \in W$.

C'est ce qu'on appelle la contrainte de localité des termes. Mais imposer cette contrainte en plus de celle de la rigidité des termes implique que tout objet dénoté par un terme doive exister dans tout monde. Cela va donc à l'encontre de ce qui motive précisément les domaines dépendant du monde.

Un second problème résulte du fait que la règle modale de nécessité et les axiomes de la logique classique permettent d'inférer la converse de la formule de Barcan (notée *CBF*), à savoir : $\Box\forall x A \supset \forall x \Box A$. Alors, à partir de la formule $\Box\forall x\exists y(y = x)$ qui est *Q1R*–valide, et de la formule (*CBF*) on peut déduire le théorème $\forall x \Box\exists y(y = x)$. La validité de cette formule entraîne que tout objet existant dans un monde w doit exister dans tous les mondes accessibles à partir de ce monde w. C'est la contrainte des mondes emboîtés, que l'on peut encore écrire :

Si wRw', alors $S(w)$ est un sous–ensemble de $S(w')$.

Si la relation R vérifie que tout monde est accessible directement ou indirectement à partir de tout autre monde (ce qui est le cas dans les structures caractérisées par le système $S5$), alors la contrainte des mondes emboîtés revient à imposer des domaines fixes.

Une solution radicale au premier problème est d'imposer que la syntaxe ne comporte pas d'autres termes que les variables et que la sémantique d'une

formule ouverte soit identique à celle de la fermeture universelle de cette formule. Ainsi, il n'est plus nécessaire de définir la fonction V pour les termes. Le très gros inconvénient de cette logique réside dans son pouvoir d'expression affaibli, conséquence de l'absence de termes.

D'autres solutions ont encore été proposées, mais qui introduisent de nouveaux problèmes (voir à ce sujet l'article de Garson (dans [Gabbay and Guenthner 84] dont la conclusion est qu'il faut préférer l'axiomatisation de la logique libre à celle de la logique classique).

4.3.3. *Désignateurs non rigides*

Dans la langue naturelle, les expressions telles que "l'homme le plus grand", "le président", etc., réfèrent naturellement à des objets différents dans les mondes différents. Afin de rendre la théorie conforme à la réalité sous-tendue par les expressions de la langue naturelle, il faut admettre que les termes puissent être des désignateurs non rigides; autrement dit les termes doivent avoir des intensions non constantes.

Les règles sémantiques qui permettent d'évaluer les formules quantifiées étant indépendantes du caractère rigide ou non des termes, les règles vues au paragraphe précédent restent valables.

La définition de désignateurs non rigides entraîne cependant l'apparition de problèmes inattendus.

On peut, par exemple, montrer que la formule :

$$\Box(t = t) \wedge E(t) \supset \exists x \Box(x = t),$$

est prouvable à partir de la règle d'instantiation universelle de la logique libre. Comme la formule $\Box(t = t)$ est également prouvable, on en déduit la validité de la formule :

$$(E\Box) \ : E(t) \supset \exists x \Box(x = t).$$

Ce que l'on ne souhaite pas. En effet, si cette formule était vraie en $w \in W$, cela voudrait dire qu'il existe un objet d dans w tel que l'extension de t soit égale à d dans tous les mondes accessibles à partir de w; il suffit alors de construire un modèle où t reçoit une extension différente dans chaque monde accessible à partir du monde w pour arriver à une discordance entre dérivabilité axiomatique et vérité sémantique.

La règle (*FUI*) d'instantiation universelle de la logique libre devrait donc être modifiée en vue de lever l'incohérence du modèle $Q1R$; telle est la raison de l'introduction des modèles $Q3$ et $Q3L$.

4.3.3.1. *Le système Q3*

On définit un modèle $Q3$ comme un modèle $Q1R$ sans la condition (*VRT*) (et donc un système axiomatique sans les règles (*RT*)). Les lois de la logique libre seraient $Q3$-valides si l'on interprétait le prédicat E de telle sorte que :

$E(t)$ est une formule vraie dans le monde w si et seulement si l'extension $V(t)(w)$ de t appartient à $S(w)$ et que $V(t)$ est une fonction constante.

Un modèle, avec ce type d'interprétation pour le prédicat E, est appelé modèle $Q3$ à termes globaux. Notons cependant que l'extension de E devrait dans ce cas contenir des intensions de terme, et non pas des objets (ou entités). Il s'ensuivrait que E serait un prédicat intensionnel, ce qui signifie que la substitution des identiques n'est plus valable pour les termes. Le système $Q3$ à termes globaux est axiomatisable au moyen de règles de généralisation et d'instantiation universelles plus complexes que les règles (*FUI*) et (*FUG*) (voir l'article de Galton dans [Gabbay and Guenthner 84]).

Il existe une autre façon d'envisager le problème.

4.3.3.2. *Le système Q3L*

Si nous ajoutons l'hypothèse que les termes sont locaux, c'est-à-dire que l'extension d'un terme dans un monde w est toujours située dans le domaine de ce monde, dans ce cas il y a moyen de revenir aux lois de quantification classiques.

Un modèle $Q3$ avec termes locaux (modèle $Q3L$) est un modèle $Q3$ qui satisfait à la condition L :

(L) : $V(t)(w) \in S(w)$ pour tout $w \in W$ et tous termes t.

Cette condition ne peut être imposée pour des systèmes avec termes rigides, car tout objet dont un terme constitue la référence devrait nécessairement exister dans tous les domaines. Cependant, au cas où les termes ne sont pas rigides, les domaines peuvent se modifier de façon correspondante.

4.3.3.3. *Application à la logique aléthique*

Une application importante de $Q3L$ se situe dans le cadre de la nécessité logique. Si $\Box$ représente la nécessité logique, alors on peut penser les mondes possibles w comme des modèles $< S(w), V(w) >$ de la logique des prédicats, chacun étant caractérisé par son domaine propre $S(w)$ et sa fonction d'assignation $V(w)$. On attend d'une fonction d'assignation $V(w)$ d'un modèle $< S(w), V(w) >$ qu'elle fournisse les extensions des termes (et des prédicats) dans le domaine $S(w)$ correspondant. Ainsi il est logique d'adopter dans ce cas des termes non rigides, des domaines dépendant du monde, une interprétation objectuelle et des termes locaux.

4.4. Domaines de quantification contenant des éléments de type intensionnel

4.4.1. *Introduction*

Les difficultés que l'on a signalées pour les sémantiques utilisant des désignateurs non rigides proviennent du fait que les termes peuvent se voir assigner des intensions alors que les domaines de quantification ne contiennent que des *intensions constantes* que l'on associe plutôt à des désignateurs rigides. Afin de lever cette absence de cohérence, on songe dès lors à admettre des intensions non constantes dans les domaines de quantification. Cela conduit à définir des *modèles conceptuels* et des *modèles substantiels*. Dans ces modèles les variables quantifiées reçoivent des intensions non constantes. Si s représente le type formel associé aux mondes possibles et que e est le type associé aux entités, alors les domaines de quantification des *modèles conceptuels* sont formés d'éléments de type $s \mapsto e$, tandis que les domaines de quantification des *modèles substantiels* sont formés d'une catégorie particulière de ces éléments, appelés *substances*.

4.4.2. *Interprétation conceptuelle*

Dans cette interprétation, le domaine de quantification est formé d'éléments de type $s \mapsto e$; intuitivement, cela est satisfaisant. En effet, si on voit chacun des mondes comme un état de l'univers à un instant donné, alors les objets sémantiques, puisqu'ils peuvent évoluer d'un monde à un autre, doivent être identifiés à des intensions de terme. Ainsi, quantifier sur un ensemble d'objets sémantiques revient formellement à quantifier sur des fonctions de W dans S. Quant à l'extension d'un terme dans un monde, elle correspond à l'état dans ce monde de l'objet dénoté par ce terme.

Dans un cadre temporel, les mondes possibles sont des *moments temporels*. Le domaine de quantification est alors constitué d'éléments qui sont des fonctions de ces moments vers les objets existant à chacun de ces moments.

4.4.2.1. *Domaines fixes; le système QC*

Un modèle conceptuel avec termes non rigides et domaines fixes, ou modèle QC, est un quintuplet $<W, R, S, QC, V>$ où le domaine de quantification QC est l'ensemble des fonctions de W dans S. Dans ce cas il est naturel de définir la fonction V de la façon suivante :

Règle QC :
La fonction V associe à une formule de la forme $\forall x A$ la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$ telle que $F(w)$ est $\mathbf{V}$ si et seulement si $V_{x/f}(A)(w)$ est $\mathbf{V}$ pour tout $f \in W \to S$,

la fonction $V_{x/f}(A)(w)$ étant la fonction identique à V sauf que l'intension donnée à x par $V_{x/f}$ est la fonction f.

Quoique le modèle conceptuel soit intuitivement proche d'une certaine réalité, il présente quelques défauts majeurs. Par exemple, il n'existe pas de système axiomatique (consistant) qui soit complet pour cette sémantique.

Ce modèle présente en plus des problèmes qui relèvent de l'intuition. Par exemple, on vérifie que, dans cette sémantique, les formules $\exists x \Box(x = t)$ et $\forall x \Box \exists y(y = x)$, dont il a été question précédemment, sont toutes deux des formules QC-valides. Rappelons que la seconde de ces formules signifie que "chaque chose (que l'on peut nommer) existe nécessairement".

4.4.2.2. *Domaines dépendant du monde; le système Q2*

Nous adoptons la même démarche que celle qui est utilisée dans le contexte de l'interprétation objet pour passer des domaines fixes aux domaines dépendant du monde. A savoir, le nouveau domaine de quantification est un ensemble dépendant du monde et constitué d'éléments de l'ancien domaine. Le nouveau modèle ainsi obtenu, appelé modèle *Q2*, est défini de la façon suivante.

Un *modèle orienté concept* (ou *modèle conceptuel*) avec termes non rigides et domaines dépendant du monde est un modèle où le domaine de quantification Q est une fonction qui associe à chaque monde $w \in W$ un sous-ensemble $C(w)$ des fonctions de W dans S.

Dans ce cas il est naturel de définir la fonction de valuation V comme suit :

> **Règle** *Q2* :
> La fonction V associe à une formule de la forme $\forall x A$ la fonction F de W dans $\{\mathbf{V}, \mathbf{F}\}$ telle que $F(w)$ est $\mathbf{V}$ si et seulement si $V_{x/f}(A)(w)$ est $\mathbf{V}$ pour tout $f \in C(w)$,

la fonction $V_{x/f}(A)(w)$ étant la fonction identique à V sauf que l'intension donnée à x par $V_{x/f}$ est la fonction f.

Les objets étant identifiés à des intensions, permettre d'avoir des objets existant dans un monde et pas dans un autre équivaut à associer à chaque monde l'ensemble des intensions qui y existent.

4.4.3. *Interprétation substantielle*

Admettons que nous donnions à la formule $\exists x \Box \exists y(y = x)$ la signification : "quelque chose doit exister dans tous les mondes possibles". On vérifie que cette formule est valide pour l'interprétation conceptuelle sous la seule condition que tous les mondes possibles aient un domaine non vide. Cela ne correspond pas à notre intuition, laquelle exige que toute intension de terme qui vérifie $\exists x \Box \exists y(y = x)$ soit "cohérente" : elle doit être la manifestation d'une chose que l'on peut reconnaître (identifier) quand on passe d'un monde à un autre,

et ne peut être une collection hétéroclite. Cette exigence est formulée avec netteté dans le slogan de Quine : "Pas d'entité sans identité". Chez Aristote, la substance première est l'individu (l'homme individuel) et la substance seconde est l'espèce (l'homme). La substance première n'est donc pas un objet quelconque, mais un objet déterminé par une essence (un complexe cohérent d'attributs qui caractérise l'espèce). Afin de rendre compte du fait que les quantifications doivent porter sur des "choses cohérentes" – que nous appelons *substances* – et non pas sur des collections hétéroclites d'objets, nous devons donc adapter les éléments qui constituent le domaine de quantification.

C'est dans ce but que l'on introduit le système QS dans lequel on considère un *un ensemble d'intensions de termes* pour chaque monde, ensemble qui servira de domaine de quantification pour ce monde.

De façon formelle on définit un *modèle substantiel relatif au monde* comme un quintuplet $<W, R, S, QS, V>$, où QS est une fonction qui assigne à chaque monde w un ensemble $\mathcal{S}(w)$ de fonctions de W vers S. (On dit que $\mathcal{S}(w)$ est l'ensemble des *substances* pour le monde w.)

La différence essentielle entre le *modèle conceptuel* et le *modèle substantiel* réside donc dans le type des termes qui constituent le domaine de quantification. Si dans le modèle conceptuel les éléments du domaine de quantification sont des fonctions $f : W \mapsto S$ qui peuvent désigner des *collections hétéroclites d'objets*, par contre dans le modèle substantiel les éléments du domaine de quantification sont des fonctions que l'on peut contraindre à ne désigner que des *substances*. A chaque monde w est associé un ensemble de *certaines* fonctions $f : W \mapsto S$ qui représentent l'ensemble des substances associées à ce monde. Cet *ensemble d'ensembles de certaines fonctions*, ou *ensemble d'ensembles de substances*, peut donc également être vu comme une fonction $F : W \mapsto W \mapsto S$, soit une fonction de type $e \mapsto e \mapsto t$.

Formellement, la règle *Q2* du modèle conceptuel est remplacée par la règle QC :

> **Règle** Qc :
> La fonction V associe à une formule de la forme $\forall x A$ la fonction F de W dans {**V**, **F**} telle que $F(w)$ est **V** si et seulement si $V_{x/f}(A)(w)$ est **V** pour tout $f \in \mathcal{S}(w)$,

la fonction $V_{x/f}(A)(w)$ étant la fonction identique à V sauf que l'intension donnée à x par $V_{x/f}$ est la fonction f.

On peut vérifier alors que la formule $\forall x \Box \exists y(y = x)$ n'est pas valide dans cette sémantique, mais qu'elle est uniquement vraie dans un monde w s'il existe une substance f dans $\mathcal{S}(w')$ pour tout monde accessible à partir de w.

Un problème qui reste à résoudre est celui de reconnaître les fonctions qui, dans chaque monde, représentent les substances. Le prédicat intensionnel E de la logique peut être utilisé à cette fin.

Chapitre 5
Logiques temporelle et épistémique quantifiées

5.1. Logique temporelle quantifiée

5.1.1. *Introduction*

La logique temporelle a été étudiée au chapitre 3 dans le cadre du formalisme propositionnel. Ce formalisme, limité dans son expressivité, rend nécessaire l'utilisation d'un langage formel plus approprié pour la prise en compte de situations "réelles". Les individus et objets du monde physique ont une existence dans un contexte temporel. Ils ont une *histoire*, ils viennent du passé et vont vers le futur en se modifiant au cours de leur existence. On est donc amené à devoir considérer, pour un instant temporel arbitraire t, l'ensemble des individus existant à cet instant t. Dans un contexte temporel, la première question qui se pose est donc celle du domaine d'existence des individus. Le *domaine de quantification* de la logique temporelle doit être construit de façon à représenter ce domaine d'existence.

Les définitions suivantes pour ce domaine sont parmi les plus usuelles :

1. L'ensemble $S(p)$ de tous les individus existant au temps présent.
2. L'ensemble $S(t)$ de tous les individus existant à un certain temps t de l'univers W.
3. L'ensemble $S^*(p) = \cup_{t \leq p} S(t)$ de tous les individus ayant existé jusqu'à présent.
4. L'ensemble $S^*(t) = \cup_{t' \leq t} S(t')$ de tous les individus ayant existé jusqu'au temps t.
5. L'ensemble S^* de tous les individus qui existent à un temps quelconque ($S^* = \cup_{t \in W} S(t)$).

Les quantificateurs universel et existentiel associés à ces domaines reçoivent

ensuite la signification suivante :

1. $\forall_p x$ signifie "pour tous les x existant au temps présent";
   $\exists_p x$ signifie "pour au moins un x existant au temps présent";
2. $\forall_t x$ signifie "pour tous les x existant au temps t";
   $\exists_t x$ signifie "pour au moins un x existant au temps t";
3. $\forall_p^* x$ signifie "pour tous les x ayant existé jusqu'à maintenant";
   $\exists_p^* x$ signifie "pour au moins un x ayant existé jusqu'à maintenant";
4. $\forall_t^* x$ signifie "pour tous les x ayant existé jusqu'au temps t";
   $\exists_t^* x$ signifie "pour au moins un x ayant existé jusqu'au temps t";
5. $\forall^* x$ signifie "pour tous les x";
   $\exists^* x$ signifie "pour au moins un x".

Chacune de ces paires de quantificateurs satisfait aux règles habituelles de la logique quantifiée; les deux formules suivantes constituent des théorèmes de cette logique :

$$\forall x A(x) \supset \exists x A(x);$$
$$\neg \forall x A(x) \equiv \exists x \neg A(x).$$

Ces quantificateurs ne satisfont donc pas à de nouvelles règles syntaxiques: la nouveauté concerne uniquement la sémantique. La signification des formules est fonction du type de quantificateur utilisé. Soit par exemple la formule :

$$\forall x(\mathit{homme}(x) \supset \mathit{mortel}(x)).$$

Cette formule signifie : "tous les hommes sont mortels", mais uniquement dans le cas où le quantificateur universel est $\forall^*$. Dans les autres cas nous aurons :

1. $\forall_p x$: tous les hommes vivant actuellement sont mortels;
2. $\forall_t x$: tous les hommes vivant au temps t sont mortels;
3. $\forall_p^*$: tous les hommes ayant vécu jusqu'à maintenant sont mortels;
4. $\forall_t^*$: tous les hommes ayant vécu jusqu'au temps t sont mortels.

Ces quantificateurs portent le nom de *quantificateurs restreints* (sous-entendu : à un certain domaine). Ces quantificateurs restreints peuvent être définis au moyen des quantificateurs classiques (ou non restreints).

1. $\forall_p$: $\forall_p x A(x) =_{def} \forall x((x \in S(p)) \supset A(x))$;
   $\exists_p$: $\exists_p x A(x) =_{def} \exists x((x \in S(p)) \wedge A(x))$.
2. $\forall_t$: $\forall_t x A(x) =_{def} \forall x((x \in S(t)) \supset A(x))$;
   $\exists_t$: $\exists_t x A(x) =_{def} \exists x((x \in S(t)) \wedge A(x))$.
3. $\forall_p^*$: $\forall_p^* x A(x) =_{def} \forall x(\forall t'((t' \leq p) \wedge (x \in S(t'))) \supset A(x))$;
   $\exists_p^*$: $\exists_p^* x A(x) =_{def} \exists x(\exists t'((t' \leq p) \wedge (x \in S(t'))) \wedge A(x))$.
4. $\forall_t^*$: $\forall_t^* x A(x) =_{def} \forall x(\forall t'((t' \leq t) \wedge (x \in S(t'))) \supset A(x))$;
   $\exists_t^*$: $\exists_t^* x A(x) =_{def} \exists x(\exists t'((t' \leq t) \wedge (x \in S(t'))) \wedge A(x))$.
5. $\forall^* x$: $\forall^* x A(x) =_{def} \forall x(\forall t(x \in S(t)) \supset A(x))$;
   $\exists^* x$: $\exists^* x A(x) =_{def} \exists x(\exists t(x \in S(t)) \wedge A(x))$.

Les quantificateurs, tels qu'ils viennent d'être définis, sont représentables en

notation typée. Cette notation incorpore un *restricteur de type*, noté T, tel que :

$$(\forall x|T(x))A(x) =_{def} (\forall x(T(x) \supset A(x))),$$

$$(\exists x|T(x))A(x) =_{def} (\exists x(T(x) \wedge A(x))).$$

Dans le cas des quantificateurs temporels restreints, nous avons :

$$\begin{array}{lcl} \forall_p x A(x) & = & (\forall x|x \in S(p))A(x); \\ & = & \forall x((x \in S(p)) \supset A(x)); \\ \exists_p x A(x) & = & (\exists x|x \in S(p))A(x); \\ & = & \exists x((x \in S(p)) \wedge A(x)). \end{array}$$

Remarque. Notons que la notation typée, où à chaque variable est associée une restriction de type, quoique fondamentalement la même que celle qui est non typée, fait apparaître une perspective légèrement différente. Plus précisément, elle établit une connexion avec une terminologie largement en usage dans les systèmes de représentation de la connaissance utilisés en intelligence artificielle.

Cette notation permet, par exemple, de représenter de façon fort simple l'opération de *réification des relations* qui élargit l'ontologie d'une théorie logique, ontologie qui exprime l'ensemble des objets dont la théorie peut parler. *Réifier* une relation consiste à la transformer en objet dont on pourra ensuite parler et que l'on pourra introduire dans une structure hiérarchisée dont le premier niveau est formé par les objets du domaine du discours.

5.1.2. *Quantification et modalités temporelles*

Rappelons tout d'abord les définitions historiques des modalités du nécessaire et du possible dans lesquelles $V_t(p)$ signifie "p est vrai au temps t".

Ecole stoïque :

$$\Diamond p = \exists t((t \geq n) \wedge V_t(p));$$
$$\Box p = \forall t((t \geq n) \supset V_t(p)).$$

Ecole mégarique :

$$\Diamond p = \exists t V_t(p);$$
$$\Box p = \forall t V_t(p).$$

Ces définitions, initialement définies en logique modale des propositions (p étant une proposition atomique), sont généralisables à la logique modale des prédicats, p devenant une forme prédicative.

Une logique temporelle quantifiée doit être capable de construire chacune des huit formules suivantes ainsi que leur équivalent en logique temporelle :

Formule	Ecole mégarique	Ecole stoïque
$\Box\forall x A(x)$:	$\forall t V_t(\forall x A(x))$	$\forall t\{(p \leq t) \supset V_t(\forall x A(x))\}$
	$? \equiv \forall t \forall x V_t(A(x))$	$? \equiv \forall t \forall x\{(p \leq t) \supset V_t(A(x))\}$
$\forall x \Box A(x)$:	$\forall x \forall t V_t(A(x))$	$\forall x \forall t\{(p \leq t) \supset V_t(A(x))\}$
$\exists x \Box A(x)$:	$\exists x \forall t V_t(A(x))$	$\exists x \forall t\{(p \leq t) \supset V_t(A(x))\}$
$\Box\exists x A(x)$:	$\forall t V_t(\exists x A(x))$	$\forall t\{(p \leq t) \supset V_t(\exists x A(x))\}$
	$? \equiv \forall t \exists x V_t(A(x))$	$? \equiv \forall t\{(p \leq t) \supset \exists x V_t(A(x))\}$
$\Diamond\forall x A(x)$:	$\exists t V_t(\forall x A(x))$	$\exists t\{(p \leq t) \wedge V_t(\forall x A(x))\}$
	$? \equiv \exists t \forall x V_t(A(x))$	$? \equiv \exists t \forall x\{(p \leq t) \wedge V_t(A(x))\}$
$\forall x \Diamond A(x)$:	$\forall x \exists t V_t(A(x))$	$\forall x \exists t\{(p \leq t) \wedge V_t(A(x))\}$
$\Diamond\exists x A(x)$:	$\exists t V_t(\exists x A(x))$	$\exists t\{(p \leq t) \wedge V_t(\exists x A(x))\}$
	$? \equiv \exists t \exists x V_t(A(x))$	$? \equiv \exists t \exists x\{(p \leq t) \wedge V_t(A(x))\}$
$\exists x \Diamond A(x)$:	$\exists x \exists t V_t(A(x))$	$\exists x \exists t\{(p \leq t) \wedge V_t(A(x))\}$

L'équivalence potentielle, notée $? \equiv$, dépend de la validité de la règle :

$$V_t(\forall x A(x)) \equiv \forall x V_t(A(x)),$$

ou de sa duale :

$$V_t(\exists x A(x)) \equiv \exists x V_t(A(x)).$$

Ces équivalences sont réalisées pour les paires de quantificateurs $\forall^*/\exists^*$, c'est-à-dire dans le cas où les domaines de quantification ne dépendent pas du monde. Les formules suivantes sont, dans ce cas, valides :

1. $\Box\forall x A(x) \equiv \forall x \Box A(x)$;
2. $\Diamond\forall x A(x) \supset \forall x \Diamond A(x)$;
3. $\Diamond\exists x A(x) \equiv \exists x \Diamond A(x)$;
4. $\exists x \Box A(x) \supset \Box\exists x A(x)$.

Démontrons par exemple la validité de la formule 2 :

$$\Diamond\forall x A(x) \supset \forall x \Diamond A(x) \Longleftrightarrow$$
$$\exists t \forall x V_t A(x) \supset \forall x \exists t V_t A(x).$$

Cette dernière expression étant une formule valide de la logique des prédicats, la formule 2 l'est également.

Si les équivalences ne sont pas réalisées, les formules 1–4 ne sont pas valides. Par exemple, on peut démontrer que ces équivalences ne sont pas réalisées pour les paires de quantificateurs $\forall_p/\exists_p$ et $\forall_p^*/\exists_p^*$ (voir l'exemple donné au § 5.1.3, figure 5.1). On peut aussi, dans certains cas, se contenter des formules 1 et 3

sous une forme "atténuée"; telles sont par exemple les formules de Barcan qui font partie de certains systèmes axiomatiques :

$\Diamond\exists x A(x) \supset \exists x \Diamond A(x)$;
$\forall x \Box A(x) \supset \Box \forall x A(x)$.

Rappelons qu'introduire les formules de Barcan comme axiomes dans un système axiomatique revient à imposer la propriété $\forall w \forall v (wRv \supset \forall x (Exv \supset Exw))$ à la relation d'accessibilité.

De même, imposer la validité de la formule $\Box\exists x A(x) \supset \exists x \Box A(x)$ équivaut à imposer la propriété $\forall w \forall v (vRw \supset \forall x (Exw \supset Exv))$. Enfin, à la validité de la formule $\Box\exists x A(x) \supset \exists x \Box A(x)$ est associée une condition d'ordre supérieur sur la relation d'accessibilité.

Considérons encore la formule suivante :

$\exists x \Box A(x) \supset \Box\exists x A(x)$.

La partie de gauche de l'implication peut s'interpréter : "Il existe un x tel que pour *cet* x, dans chaque monde possible $A(x)$ est vrai", tandis que la partie de droite se lit alors : "Dans chaque monde possible il existe un x – dépendant du monde – tel que $A(x)$ est vrai". L'énoncé attaché à la partie de gauche est évidemment plus restrictif que celui qui est attaché à la partie de droite, ce qui justifie le sens de l'implication.

Considérons enfin la formule :

$\Diamond\forall x A(x) \supset \forall x \Diamond A(x)$.

La partie de gauche de cette implication peut s'interpréter : "Il existe un monde tel que dans *ce* monde $\forall x A(x)$ est vrai", tandis que la partie de droite se lit : "Pour chaque x il existe un monde – dépendant de x – où $A(x)$ est vrai". A nouveau, l'énoncé attaché à la partie de gauche est plus restrictif que celui attaché à la partie de droite.

5.1.3. *Version quantifiée de logiques normales*

La construction d'une version quantifiée, notée QK_t, de la logique K_t (plus petite logique normale) est réalisée en ajoutant au langage K_t les quantificateurs et le prédicat d'existence E de la logique libre, défini précédemment comme suit :

$$V(E(x))(t) \Longleftrightarrow x \in S(t).$$

La valeur sémantique de $E(x)$ au temps t vaut donc **V** si et seulement si x est un existant au temps t, c'est-à-dire si et seulement si x appartient à l'ensemble $S(t)$.

Le quantificateur universel de la logique QK_t porte sur tous les individus pouvant exister à un temps quelconque. Dans les cas – tel par exemple celui de la logique K_b – où les séries temporelles ne sont pas linéaires, le domaine de quantification doit contenir tous les individus existant sous n'importe quelle possibilité temporelle.

Il s'agit ensuite de définir les quantificateurs restreints en terme de quantificateurs non restreints, soit :

$$\forall_p x A(x) =_{def} \forall x(E(x) \supset A(x)),$$
$$\forall_p^* A(x) =_{def} \forall x[P](E(x) \supset A(x)) \wedge \forall_p x A(x).$$

Les quantificateurs restreints diffèrent des quantificateurs non restreints en ce qu'ils ne permutent pas avec les opérateurs temporels. Par conséquent les formules :

$$\forall x[F]A(x) \equiv [F]\forall x A(x),$$
$$\forall x[P]A(x) \equiv [P]\forall x A(x),$$

qui sont valides pour les quantificateurs non restreints, ne le sont pas pour les quantificateurs restreints. Par exemple, la formule :

$$\forall_p x[F]A(x) \supset [F]\forall_p x A(x),$$

est fausse au temps t_1 dans le modèle de la figure 5.1. En effet, tous les individus qui existent au temps t_1 (à savoir a) seront A dans le futur, c'est-à-dire au temps t_2; donc l'antécédent de la formule est vrai. Mais il est faux au temps t_1 qu'il sera vrai au temps t_2 que tous les x qui existeront alors (à savoir a et b) seront A. Donc le conséquent est faux.

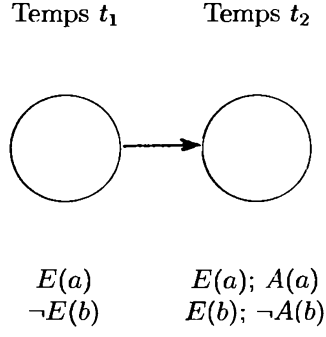


Figure 5.1 *Exemple d'univers composé de deux instants temporels*

Le même modèle rend également fausse au temps t_1 la formule :

$$\forall_p^* x[F]A(x) \supset [F]\forall_p^* x A(x).$$

Par ailleurs, certains schémas qui sont des axiomes pour la logique des prédicats ne le sont plus dans le cadre de logiques modales utilisant des quantificateurs restreints; tel est par exemple le cas des schémas :

$$\forall_p x A(x) \supset A(y);$$
$$\forall_p^* x A(x) \supset A(y).$$

La raison est due au fait que les individus n'existent pas en tous temps.

5.1.4. *Changement temporel, identité, loi de Leibniz*

Le fait que quelque chose puisse se modifier au cours du temps implique que quelque chose soit *la même chose* à deux instants différents : *il n'y a pas de changement sans perdurance.* Ceci peut encore s'énoncer : "L'objet qui au temps t a la propriété A est le même objet qui au temps t' n'a pas cette propriété". Ce qui peut se formuler :

$$A(x) \qquad \neg A(y) \qquad x = y$$

Mais cette formulation est inconsistante avec la loi de Leibniz qui s'énonce :

$$x = y \Longleftrightarrow \forall X(X(x) \equiv X(y)).$$

Il faut donc modifier la formulation de Leibniz afin d'y incorporer une composante temporelle; la nouvelle situation ainsi créée peut être décrite comme suit :

$$A_t(x) \qquad \neg A_t(y) \qquad x = y$$

La loi de Leibniz peut donc être maintenue, pour autant que l'on incorpore une composante temporelle dans les noms de prédicats. La "propriété A" d'un objet doit être remplacée par sa "propriété A au temps t".

Une autre possibilité est de restaurer la loi de Leibniz sous une forme temporalisée, c'est-à-dire :

$$x = y \Longleftrightarrow \forall X \forall t(V_t(X(x)) \equiv V_t(X(y))).$$

Dans le cas d'un *futur arborescent*, cette forme temporalisée doit être modifiée dans le sens suivant :

$$x = y \Longleftrightarrow \exists t' \forall X \forall t\{(t \leq t') \supset (V_t(X(x)) \equiv V_t(X(y)))\},$$

qui indique que des individus ne peuvent être reconnus comme identiques que si leurs histoires coïncident jusqu'au point de branchement [Rescher and Urquhart 71].

5.1.5. *Logique modale quantifiée et temps arborescent*

Notons $<x>$ l'ensemble des x qui sont indiscernables selon le critère de Leibniz dans un modèle temporel arborescent. Dans ce contexte on introduit les modalités de la façon suivante :

$$\Diamond A(x) =_{def} \exists y((y \in <x>) \wedge A(y)).$$

Ceci signifie qu'il est possible que x satisfasse A pour autant qu'il existe une x-variante de x qui, prenant une valeur dans l'ensemble $<x>$, rende la formule A vraie. L'opérateur universel est défini par dualité :

$$\Box A(x) =_{def} \forall y((y \in <x>) \supset A(y)).$$

En accord avec ces définitions, nous avons :

$$\begin{aligned} \Diamond[V_t(A(x))] &= \exists y[(y \in <x>) \wedge V_t(A(y))]; \\ \Box[V_t(A(x))] &= \forall y[(y \in <x>) \supset V_t(A(y))]. \end{aligned}$$

Ces égalités peuvent encore être écrites de la façon suivante :

$$\begin{aligned} V_t[\Diamond(A(x))] &= V_t\{\exists y[y \in <x> \wedge A(y)]\}, \\ &?= \exists y[y \in <x> \wedge V_t(A(y))]; \\ V_t[\Box(A(x))] &= V_t\{\forall y[y \in <x> \supset A(y)]\}, \\ &?= \forall y[y \in <x> \supset V_t(A(y))]. \end{aligned}$$

L'égalité notée ? = dépend de la validité de la formule qui rend compte de la commutativité du quantificateur Q et de l'opérateur V_t :

$$V_t[(Qx)A(x)] \equiv QxV_t(A(x)).$$

Voyons ce que deviennent dans ce contexte certaines expressions (de la logique modale) vraies sous les conditions (vues ci-dessus) de commutativité de l'opérateur V_t avec les quantificateurs universel et existentiel; tout d'abord :

$$\Diamond\forall xA(x) \supset \forall x\Diamond A(x),$$

devient :

$$\Diamond\forall xA(x) \supset \forall x\exists y(y \in <x> \wedge A(y)).$$

De même :

$$\Box\forall xA(x) \supset \forall x\Box A(x),$$

devient :

$$\Box\forall xA(x) \supset \forall x\forall y(y \in <x> \supset A(y)).$$

5.2. Logique épistémique quantifiée

Nous nous limitons à attirer l'attention du lecteur sur certaines spécificités de la logique épistémique quantifiée. Tout d'abord, cette logique est généralement basée sur l'utilisation de *désignateurs non rigides*. Considérons un exemple désormais classique. Si t_1 est un terme désignant *l'étoile du matin*, que t_2 est un terme désignant *l'étoile du soir*, nous savons que $t_1 = t_2$, étant donné que ces expressions représentent toutes deux la planète Vénus. Mais les anciens, qui observaient une étoile qui brillait le matin, et une étoile qui brillait le soir, ne savaient pas qu'il s'agissait d'une même étoile, qui, par ailleurs, était une planète. Si $[K_i]$ représente la connaissance d'un ancien, la formule suivante ne peut donc être valide :

$$(t_1 = t_2) \supset [K_i](t_1 = t_2).$$

Or, cette formule est dérivable de l'axiome de la logique du premier ordre :

$$(t_1 = t_2) \supset (A(t_1) \equiv A(t_2)),$$

qui ne peut donc plus être incorporé dans un système axiomatique formalisant la connaissance et utilisant des termes non rigides. La dérivation procède comme suit. Dans l'axiome précité de la logique du premier ordre, une substitution convenable sur A permet d'obtenir :

$$(t_1 = t_2) \supset ([K_i](t_1 = t_1) \equiv [K_i](t_1 = t_2)).$$

Des transformations triviales donnent successivement :

$$(t_1 = t_2) \supset ([K_i](t_1 = t_1) \supset [K_i](t_1 = t_2))$$
$$[K_i](t_1 = t_1) \supset ((t_1 = t_2) \supset [K_i](t_1 = t_2)).$$

Comme les anciens sont des êtres rationnels, on peut poser $[K_i](t_1 = t_2)$, d'où l'on tire la formule visée par *modus ponens*.

Une solution à ce problème est de restreindre certains axiomes de la logique à des formules contenant exclusivement des variables et non plus des termes. Une solution de ce type avait déjà été envisagée dans l'axiomatisation de systèmes dont la sémantique est basée sur des mondes emboîtés. Dans le cas de la logique épistémique quantifiée, la solution au problème de la sémantique des termes non rigides consiste à restreindre deux axiomes de la logique du premier ordre de façon à ce que n'apparaissent plus de termes dans la portée d'un opérateur modal. En sus de l'axiome qui vient d'être mentionné, on utilisera encore la version restreinte (sans termes) de l'axiome :

$$A(y) \supset \exists x A(x).$$

Un système axiomatique pour une logique épistémique est représentable sous forme de théorème [Fagin et al. 95] :

Théorème 5.1. Soit $\mathcal{L}$ une logique modale propositionnelle qui est adéquate et complète par rapport à un ensemble de structures de Kripke.

On obtient une logique épistémique quantifiée, qui est adéquate par rapport à une théorie de la connaissance du premier ordre en ajoutant à $\mathcal{L}$ les ingrédients suivants :

- les axiomes *PL* de la logique des prédicats dans leur version restreinte pour :
$$(t_1 = t_2) \supset (A(t_1) \equiv A(t_2));$$
$$A(t) \supset \exists x A(x).$$
- la *formule de Barcan* (au cas où on se situe dans un contexte où les domaines sont fixes) :
$$\forall x([K_i]A) \supset [K_i](\forall x A).$$
- l'axiome de connaissance de l'inégalité :
$$(x_1 \neq x_2) \supset [K_i](x_1 \neq x_2).$$

Chapitre 6
Théorie des modèles. Première partie

6.1. Introduction

On peut voir dans la logique modale la discipline qui traite des raisonnements dont le lexique logique comporte, en plus des connecteurs et des quantificateurs, des expressions comme "nécessairement", "possiblement", "il sera le cas que", "toujours dans le passé", etc. La logique modale est alors un *prolongement* ou une *extension* de la logique classique.

Dans cette perspective, la logique modale pourrait être vue comme une analyse du langage modal et des systèmes formels (procédures de preuve, règles d'inférence) qu'on peut construire à partir de ce langage. Cette perspective est essentiellement "syntaxique".

Un autre point de vue sur la logique modale consiste à voir en celle-ci la discipline qui prend pour objet les structures de transition $<W, R_1, \ldots, R_n>$, où W est un ensemble d'états (mondes possibles, moments du temps, états de connaissance), et où les R_i sont des relations d'"accessibilité".

Ces structures, introduites à la fin des années cinquante, ne sont pas simplement un outil commode pour élaborer une sémantique. Comme le souligne S. Popkorn, "elles apparaissent à plusieurs endroits en mathématique et en informatique. Les ordres partiels, les relations d'équivalence, les graphes, les automates et les algèbres de processus sont autant d'exemples de structures de transition" [Popkorn 94], p. 25.

Les deux points de vue que nous venons d'évoquer ne s'excluent pas. Ils sont complémentaires. Nous n'avons pas à choisir l'un aux dépens de l'autre.

Le premier chapitre de théorie des modèles traitera des quatre sujets suivants qui sont, comme on le verra plus loin, étroitement liés :

1. La notion de *validité.*
2. La relation de *conséquence.*
3. Le théorème de la *déduction* (dans son acception sémantique).
4. La preuve de *complétude* obtenue par la méthode du modèle canonique.

La logique modale propositionnelle n'est pas seulement un complément à la logique non modale du premier ordre. Elle peut aussi être une alternative à cette dernière.

Certaines propriétés de la relation R dans les structures de transition $< W, R >$ peuvent être exprimées à la fois par une formule de la logique du premier ordre et par une formule de la logique modale propositionnelle (voir § 2.2.5). C'est le cas, par exemple, de la transitivité qui s'exprime, en logique du premier ordre, par :

(1) $\forall x \forall y \forall z \, [(xRy \wedge yRz) \supset xRz]$;

et en logique modale par :

(2) $\Box p \supset \Box\Box p$.

Comment s'assurer que l'une de ces deux formules correspond bien à l'autre ? C'est à répondre à cette question que s'emploie la *théorie de la correspondance.*

Loin d'être des variantes notationnelles l'une de l'autre, ces deux formulations ont des propriétés très différentes : le système modal formé en ajoutant à l'axiome modal de base K l'axiome $\Box p \supset \Box\Box p$ est décidable tandis que la logique du premier ordre n'est que semi-décidable. La logique modale a donc de bonnes propriétés (telle que la décidabilité) qui manquent à la logique du premier ordre mais, en revanche, elle est moins expressive que celle-ci.

Un des défis que le logicien-informaticien s'efforcera de relever sera de rendre la logique modale plus expressive – comme le fait P. Wolper dans [Wolper 83] – tout en préservant la propriété de décidabilité, et même davantage, en préservant la décidabilité par un algorithme polynomial.

C'est la préoccupation pour la décidabilité qui motive la production du modèle fini à partir du modèle canonique au moyen de l'opération de filtration.

La seconde partie de la théorie des modèles traitera des sujets suivants qui seront développés au chapitre 7 :

1. *Filtration* et *modèle.*
2. La théorie de la *correspondance.*
3. Résultats de *préservation.*

Les deux premiers sujets ont été précisés plus haut. La troisième question peut être explicitée en ces termes : "Quelles sont les conditions nécessaires et suffisantes pour qu'une formule du premier ordre soit équivalente à une formule modale propositionnelle ?"

6.2. Affinement de la notion de conséquence en logique modale

L'une des tâches principales de la logique est de *donner une description syntaxique de la relation de conséquence logique.* On accomplit ceci *en construisant un système formel qui contrôle la relation de dérivabilité dont les propriétés opératoires doivent mimer la relation sémantique de conséquence logique* [Popkorn 94]. Le point culminant de ce programme est le théorème *d'adéquation* affirmant que si une formule B est *dérivable* d'un ensemble Λ de formules, ce qui se note $\Lambda \vdash B$, elle est une *conséquence logique* de cet ensemble, ce qui se note $\Lambda \models B$ et, *réciproquement*, que si la formule B est une conséquence logique de cet ensemble, elle en est dérivable. Le but de ce chapitre est d'introduire les notions préliminaires nécessaires aux preuves de correction et de complétude de la logique modale propositionnelle standard. Dans le chapitre suivant, on exposera la preuve de complétude du système K par la méthode du modèle canonique.

La notion syntaxique de dérivabilité est relative à un système formel, constitué d'axiomes, de règles et de définitions. Les axiomes de plusieurs systèmes, ainsi que les règles de dérivation, encore appelées *règles d'inférence*, ont été présentés dans les chapitres qui précèdent. Nous n'y reviendrons pas. Il importe, en revanche, que nous examinions de près la notion de *conséquence logique* car cette notion fondamentale est beaucoup plus complexe en logique modale qu'elle ne l'est en logique classique.

Fitting a bien mis en évidence cette complexité accrue dans un passage que nous citons *in extenso*[1] : "La conséquence sémantique en logique classique est facile à caractériser : B est une conséquence d'un ensemble Λ si et seulement si B est vrai dans tout modèle dans lequel les membres de Λ sont vrais. En logique modale, les choses sont plus compliquées. Par exemple, nous pourrions dire que B est une conséquence de Λ si, pour chaque modèle à base de mondes possibles, B est vrai dans chaque monde où les membres de Λ sont vrais. Un moment de réflexion suffit pour comprendre que cela revient à concevoir les membres de Λ comme des vérités supposées, non comme des *vérités logiques* posées au départ. Nous pourrions dire également que B est une conséquence de Λ si B est valide dans chaque modèle dans lequel tous les membres de Λ sont valides[2]. Cette deuxième définition correspond à l'adoption de la conception selon laquelle les membres de Λ sont des *vérités* de logique. Il existe encore un autre niveau. Au lieu de travailler au niveau des mondes ou des modèles, nous pourrions travailler au niveau des structures" [Fitting 93], p. 380.

Fitting distingue trois notions de conséquence logique : la *conséquence locale* définie en termes de vérité relative à un monde, la *conséquence globale* définie en termes de validité dans un modèle, et la *conséquence globale* définie en termes

[1] Les symboles ont été modifiés pour uniformiser la notation de [Chagrov and Zakharyaschev 97].

[2] Rappelez-vous que *valide* signifie vrai dans chaque monde possible.

de validité dans une structure. Pour comprendre la portée de cette tripartition, il faut commencer par *comparer les différentes notions de validité* qui ont cours en logique modale et préciser l'enjeu qui s'attache au choix de chacune d'elles.

6.3. Validité en logique modale

6.3.1. *Vérité et validité*

La notion de vérité utilisée en logique modale est relative non seulement à un modèle – comme c'est le cas pour la logique classique – mais aussi à un monde. Une notion de vérité plus générale sera introduite au § 6.3.2, celle de vérité dans tous les mondes d'un modèle. Certains auteurs ([Hughes and Cresswell 84], [Fitting 93]) emploient l'expression de "validité dans un modèle" pour désigner la vérité dans tous les mondes d'un même modèle. Nous adopterons leur terminologie.

6.3.2. *Validité dans un modèle*

Soit un modèle $\mathcal{M} = <W, R, V>$, c'est-à-dire une structure $\mathcal{F} = <W, R>$ munie d'une valuation V. On dit que la formule A est valide dans le modèle $\mathcal{M}$ si A est vrai quel que soit le monde dans lequel on l'évalue, ou plus simplement, si A est vrai dans tous les mondes du modèle (voir § 2.2.3).

Exemple. Soit le modèle $\mathcal{M}_0 = <W, R, V>$ où $W = \{w_1, w_2\}$, $w_1 R w_2$ et $V(p, w_1) = \mathbf{V}, V(p, w_2) = \mathbf{V}$. Dans ce modèle, $\Box p \supset p$ est valide. En effet, puisque le conséquent du conditionnel p est vrai quel que soit le monde dans lequel on l'évalue, le conditionnel est lui-même vrai dans tous les mondes.

6.3.3. *Validité dans une classe de modèles*

On dit que la formule A est valide dans une classe de modèles $C(\mathcal{M})$ si A est valide dans tous les modèles de la classe $C(\mathcal{M})$, c'est-à-dire vraie quel que soit le monde et quel que soit le modèle.

La validité dans un modèle est un cas particulier de la validité dans une classe de modèles. C'est le cas où la classe $C(\mathcal{M})$ est un singleton.

La notion de validité dans une classe de modèles permet d'isoler une collection de formules logiquement intéressantes. En effet, la collection des formules valides dans une classe de modèles possède les quatre propriétés suivantes [Fitting 93], p. 376 :

1. Elle contient toutes les *tautologies*.
2. Elle contient l'axiome K, c'est-à-dire : $\Box(A \supset B) \supset (\Box A \supset \Box B)$.
3. Elle est fermée sur la règle du *modus ponens*.

4. Elle est fermée sur la règle de *nécessité*.

Bornons-nous à prouver les propositions 2 et 4.

On démontre la proposition 2 par l'absurde. Pour rendre K faux, il faudrait que $A \supset B$ soit vrai dans tous les mondes accessibles, que A soit également vrai dans tous les mondes accessibles, mais qu'il y ait un monde accessible où B n'est pas vrai. Cela est exclu par le fait que les mondes possibles respectent la logique classique. On ne peut donc avoir un monde w tel que w rende vrai $A \supset B$ ainsi que A, mais rende faux B (ce qui se note $\models_w A \supset B, \models_w A$ et $\not\models_w B$).

On prouve la proposition 4 par un argument *a fortiori* : si A est valide, c'est-à-dire vrai dans tous les mondes du modèle, A est *a fortiori* vrai dans tous les mondes accessibles, d'où $\Box A$ est valide.

La notion de validité dans un modèle possède cependant une mauvaise propriété logique. La collection des formules valides dans un modèle isolé $\mathcal{M}$ n'est pas fermée sur la règle de substitution uniforme.

Nous introduisons tout d'abord une représentation graphique des modèles. Les cercles noirs désignent les mondes, les flèches désignent les relations d'accessibilité. Les variables propositionnelles reçoivent implicitement la valeur **V**. On convient de représenter **F** par la négation. On voit aisément que $\Box p \supset p$

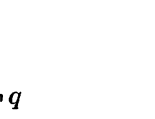


Figure 6.1 *Exemple de représentation graphique*

est valide dans le modèle de la figure 6.1 mais que $\Box q \supset q$ ne l'est pas. En effet, cette formule est fausse dans le monde w_1. Or $\Box q \supset q$ est obtenu par substitution uniforme sur $\Box p \supset p$ ([Hughes and Cresswell 84], p. 13).

Pour obtenir la fermeture de $C(\mathcal{M})$ sur la règle de substitution uniforme, il faut introduire une notion plus générale de validité, celle de *validité dans une structure*.

6.3.4. *Validité dans une structure*

Soit une collection de modèles $C(\mathcal{M})$ basés sur la structure $\mathcal{F} = <W, R>$, la formule A est dite valide dans la structure $\mathcal{F}$ si elle est vraie dans tous les modèles $\mathcal{M}$ de $C(\mathcal{M})$ pour toutes les valuations V.

Une valuation assigne à tous les atomes propositionnels la classe des mondes dans lesquels ils sont vrais. Parler de "toutes les valuations", c'est quantifier sur des ensembles de mondes (des sous-ensembles de W).

La notion de validité dans une structure, contrairement à la notion de validité dans un modèle, relève ainsi de la logique du deuxième ordre.

Des structures différentes donnent lieu à des classes de modèles différentes et donc à des notions de validité différentes. Une formule est valide dans la classe des modèles transitifs si elle est valide dans tous les modèles construits en partant d'une structure $\mathcal{F} = <W, R>$, où la relation R est transitive et en y ajoutant une valuation V quelconque. Si une formule A est valide pour toutes les valuations V du modèle $<W, R, V>$ basé sur la structure $<W, R>$ où R est transitive, la formule A' obtenue à partir de A au moyen de la règle de substitution uniforme sera également valide.

La notion de validité qu'on vient de définir est une notion relative. Une formule peut être valide dans une classe C' de modèles basée sur la structure $\mathcal{F}'$ et non valide dans une classe C'' basée sur la structure $\mathcal{F}''$. Par exemple, il y a des formules qui sont valides dans tous les modèles transitifs, mais qui ne le sont pas dans tous les modèles réflexifs. C'est le cas de la formule $\Box p \supset \Box\Box p$. Il suffit pour le voir de considérer le modèle réflexif de la figure 6.2 dans lequel la formule précitée est fausse au point w_1 (c'est-à-dire dans le monde w_1).

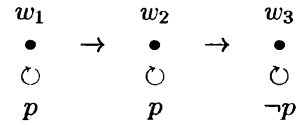


Figure 6.2 *Modèle réflexif*

6.3.5. *Valuation et substitution*

On notera que la substitution, comme la valuation, est une fonction. L'une et l'autre sont définies récursivement. La substitution σ est une fonction qui prend ses arguments dans l'ensemble des atomes propositionnels et ses valeurs dans les formules; elle est définie par la récursion suivante [Popkorn 94], p. 18 :

$$
\begin{aligned}
\sigma(p); & \\
(\neg A)\sigma &= \neg A\sigma; \\
(A \wedge B)\sigma &= (A\sigma \wedge B\sigma); \\
(A \vee B)\sigma &= (A\sigma \vee B\sigma); \\
(A \supset B)\sigma &= (A\sigma \supset B\sigma); \\
(\Box A)\sigma &= \Box A\sigma.
\end{aligned}
$$

La valuation V est une fonction qui prend ses arguments dans les atomes propositionnels et ses valeurs dans $\mathcal{P}(W)$, c'est-à-dire dans les parties de W, ou, plus précisément, dans $\mathcal{P}(W) \mapsto \{\mathbf{V}, \mathbf{F}\}$. En effet, évaluer une formule, c'est spécifier l'ensemble des mondes possibles dans lesquels elle est vraie. Ce

qui revient à spécifier pour chaque monde si elle est vraie ou fausse dans ce monde.

Comprendre une formule c'est être capable de l'évaluer. Cette conception de la compétence sémantique n'efface nullement la différence entre la *connaissance linguistique* et la *connaissance factuelle.* Pour avoir une connaissance sur le monde réel, il ne suffit pas de savoir quel ensemble de mondes possibles rend vraie une formule (ou – en langue naturelle – un énoncé). Il faut savoir *en plus* si le monde *réel* est contenu dans cet ensemble de mondes possibles.

Montrons à présent que la collection des formules *valides dans une structure* (c'est-à-dire dans un modèle dont on fait varier la valuation V *ad libitum*) est fermée sur l'opérateur de substitution.

Soit le modèle $\mathcal{M} = <W, R, V>$; on commence par construire un nouveau modèle $\mathcal{M}^*$ basé sur la même structure $\mathcal{F} = <W, R>$ dans lequel la valeur de p dans un monde w pour la valuation V^* est la même que la valeur du substitut de p dans le monde w pour la valuation V; ceci se note formellement $\mathcal{M}^* \models_w p \Leftrightarrow \mathcal{M} \models_w p\sigma$. On étend ensuite aux formules A de complexité quelconque ce qui, au départ, valait seulement pour les atomes. On procède par induction (l'hypothèse inductive sera notée *H.I.*).

1. Soit $A = \neg B$.

$$\begin{aligned}
\mathcal{M}^* \models_w B &\Leftrightarrow \mathcal{M} \models_w B\sigma \qquad (H.I.)\\
\mathcal{M}^* \not\models_w B &\Leftrightarrow \mathcal{M} \not\models_w B\sigma\\
\mathcal{M}^* \models_w \neg B &\Leftrightarrow \mathcal{M} \models_w \neg B\sigma\\
\mathcal{M}^* \models_w \neg B &\Leftrightarrow \mathcal{M} \models_w (\neg B)\sigma\\
\mathcal{M}^* \models_w A &\Leftrightarrow \mathcal{M} \models_w A\sigma
\end{aligned}$$

2. Soit $A = B \vee C$.

$$\begin{aligned}
\mathcal{M}^* \models_w B &\Leftrightarrow \mathcal{M} \models_w B\sigma \qquad (H.I.)\\
\mathcal{M}^* \models_w C &\Leftrightarrow \mathcal{M} \models_w C\sigma \qquad (H.I.)\\
\mathcal{M}^* \models_w B \vee \mathcal{M}^* \models_w C &\Leftrightarrow \mathcal{M} \models_w B\sigma \vee \mathcal{M} \models_w C\sigma\\
\mathcal{M}^* \models_w B \vee C &\Leftrightarrow \mathcal{M} \models_w B\sigma \vee C\sigma\\
\mathcal{M}^* \models_w B \vee C &\Leftrightarrow \mathcal{M} \models_w (B \vee C)\sigma\\
\mathcal{M}^* \models_w A &\Leftrightarrow \mathcal{M} \models_w A\sigma
\end{aligned}$$

3. Soit $A = \Box B$.

Prouvons que $\mathcal{M}^* \models_w \Box B \Rightarrow \mathcal{M} \models_w (\Box B)\sigma$.
Supposons que $\forall w : \mathcal{M}^* \models_w \Box B$.
Par la définition de $\Box$, $\forall w'(wRw' \Rightarrow \mathcal{M}^* \models_{w'} B\sigma)$,

$\mathcal{M}^* \models_{w'} B\sigma \Leftrightarrow \mathcal{M} \models_{w'} B\sigma$, par hypothèse inductive.
Donc $\forall w'\ (wRw' \Rightarrow \mathcal{M} \models_{w'} B\sigma)$, par échange des équivalents;
donc par définition de $\Box$: $\mathcal{M} \models_w \Box B\sigma$,
et $\mathcal{M} \models_w (\Box B)\sigma$.
D'où $\mathcal{M}^* \models_w A \Rightarrow \mathcal{M} \models_w A\sigma$.
La démonstration de $\Leftarrow$ est analogue.

En conclusion, nous pouvons énoncer le lemme suivant :

Lemme 6.1. $\forall A : \mathcal{M}^* \models_w A \Leftrightarrow \mathcal{M} \models_w A\sigma$.

Théorème 6.2. Si A est $\mathcal{F}$-valide, c'est-à-dire valide dans la structure $\mathcal{F}$ (ou dans la classe de structures $C(\mathcal{F})$), alors la formule $A\sigma$ obtenue par substitution uniforme sur A est également valide [Fitting 93], p. 376.
Démonstration. On prouve la contraposée. Supposons que $A\sigma$ ne soit pas valide dans $\mathcal{F}$, alors il existe un modèle $\mathcal{M}$ et un monde w tel que $\mathcal{M} \not\models_w A$. Dès lors, en vertu du lemme 6.1, il existe un modèle $\mathcal{M}^*$ et un monde w tel que $\mathcal{M}^* \not\models_w A$. Donc A n'est pas $\mathcal{F}$-valide.

6.3.6. *Utilité de conserver la notion de validité dans une classe de modèles*

On notera que la notion de validité dans une classe de *modèles*, par opposition à la notion de validité dans une classe de *structures*, conserve son utilité dans des cas très particuliers.

Imaginons, par exemple, que nous voulions représenter une conception du temps cosmologique selon laquelle le monde a un commencement, mais pas de fin. On pourrait penser que pour représenter cette conception du temps, il nous suffit d'adopter une structure dans laquelle W est l'ensemble infini des moments du temps associé aux nombres naturels et R la relation de l'avant à l'après. Ce n'est pas le cas. La structure ainsi définie est trop pauvre. Elle ne suffit pas à rendre compte du fait qu'à chaque moment certaines propositions contingentes (comme *"César passe le Rubicon"*) acquièrent une valeur de vérité et que celle-ci, une fois acquise, demeure à jamais inchangée. Ce qu'on exprime communément en disant que le passé est inaltérable. Thomason a démontré qu'aucune contrainte nouvelle imposée à la *structure* ne suffit à capter cette propriété du temps. La seule manière d'y parvenir, c'est d'imposer des restrictions aux *valuations* admissibles, restrictions qui excluent les assignations instables de valeurs de vérité aux atomes propositionnels [Hughes and Cresswell 84]. *Ici la notion de validité dans une classe de modèles ne peut se réduire à la notion de validité dans une structure.*

Le résultat de *Thomason* était inattendu. Bien qu'on n'arrive pas à exprimer le caractère irrévocable du passé uniquement en imposant des contraintes à la structure $\mathcal{F} = < W, R >$ sans en imposer aux valuations, on réussit à capter une idée très proche. Prenons la version temporelle de l'axiome de *McKinsey*

$\Box\Diamond p \supset \Diamond\Box p$, c'est-à-dire $GF \Rightarrow FG$, où G signifie "toujours dans le futur" et F signifie "au moins une fois dans le futur". Cet axiome dit que "toute information vraie quelque part dans le futur arrivera à un moment donné du futur à être toujours vraie à partir de ce moment" [Blackburn et al.].

6.4. Relations de conséquence logique

6.4.1. *La conséquence logique en logique non modale*

La notion de conséquence logique dans le calcul non modal des propositions est facile à définir.

Définition 6.1. B est une conséquence logique de Λ ($\Lambda \models B$), si et seulement si B est vrai dans tous les modèles dans lesquels tous les membres de Λ sont vrais.

Ce concept convient pour les formules fermées. Dans le calcul des prédicats, on peut avoir des formules ouvertes telles que "Fx", "Gxv", "$\exists y\ Hxy$". Contrairement aux formules fermées qui sont vraies ou fausses *absolument*, les formules ouvertes sont vraies ou fausses *relativement* à l'assignation d'une valeur à leurs variables ouvertes.

6.4.2. *Les variables au sens conditionnel et au sens de la généralité*

La présence des variables libres complique la définition de la conséquence logique. En effet, les variables libres peuvent être comprises de deux manières : au sens *conditionnel* et au sens de la *généralité* [Kleene 71], p. 110.

Considérons les deux égalités suivantes :

(6.1) $\quad (x-2)^2 + 4x - 8 = 0;$

(6.2) $\quad x(3+4) = 3x + 4x.$

A partir de la première, on peut faire la dérivation suivante :

$x^2 + 4 - 4x + 4x - 8 = 0;$

$x^2 - 4 = 0;$

$(x-2)(x+2) = 0;$

$x - 2 = 0 \ \vee \ x + 2 = 0;$

$x = 2 \ \vee \ x = -2;$

$\left[(x-2)^2 + 4x - 8 = 0\right] \supset \left[x - 2 = 0 \ \vee \ x + 2 = 0\right];$

$\forall x \left[\left[(x-2)^2 + 4x - 8 = 0\right] \supset \left[x = 2 \ \vee \ x = -2\right]\right].$

La disjonction des égalités dans le conséquent est vraie d'un nombre à condition que ce dernier vérifie l'antécédent.

A partir de la deuxième, on peut faire la dérivation suivante :

$2(3 + 4) = 2.3 + 2.4$;

$\forall x\,[x(3 + 4) = 3x + 4x] \supset [2(3 + 4) = 2.3 + 2.4]$.

On notera la différence de portée du quantificateur universel dans la conclusion des deux dérivations.

Dans l'équation (6.1), la variable reçoit une valeur dans la première formule (prise comme "condition") et conserve cette même valeur pendant toute la dérivation qui commence par cette formule et se termine au moment où la condition est déchargée. On dit que la variable x dans l'équation (6.1) est prise au *sens conditionnel*.

Dans l'équation (6.2), tout se passe comme si un quantificateur universel implicite liait les occurrences de la variable x. Ici, la variable universelle est autorisée à varier en parcourant tous les membres du domaine. En revanche, le contexte de substitution n'est plus l'ensemble de la dérivation, mais seulement la formule dans laquelle la variable libre apparaît. On dit que la variable x dans l'équation (6.2) est prise au *sens de la généralité*.

Cette distinction entre deux sens de la variable donne naissance à deux concepts de conséquence dans la logique du premier ordre : la conséquence reposant sur les variables prises au sens conditionnel et la conséquence reposant sur les variables prises au sens de la généralité. En logique classique, c'est la première acceptation du mot "conséquence" qui est standard. Pour cette raison, on ne lui donne pas de nom particulier. On la représente par $\models$. Pour désigner la seconde, on parle explicitement de "conséquence au sens de la généralité" et on la représente par le symbole $\models^x$ surmonté d'une variable libre. La variable libre est censée lier universellement les prémisses.

En logique modale propositionnelle, on retrouvera une distinction qui correspond exactement à la distinction entre $\models$ et $\models^x$ en logique du premier ordre : la distinction entre *conséquence locale* et *conséquence globale*; mais cette fois, les deux sens de "conséquence" seront utilisés et pourront même être utilisés au sein d'une même déduction. Ce qui demandera un aménagement du formalisme.

Pour comprendre la distinction entre $\models$ et $\models^x$ en logique du premier ordre, il faut se rappeler la distinction entre l'interprétation des formules ouvertes et des formules fermées I, p. 300-305. Les formules ouvertes sont satisfaites *par* (vraies *de*) toute, quelque ou aucune assignation de valeurs à leurs variables libres. Les formules fermées sont simplement vraies ou fausses dans un modèle (c'est-à-dire pour un domaine et une interprétation). La vérité et la fausseté sont, à leur tour, définies à partir de la notion de satisfaction. Une formule est vraie si elle est satisfaite pour toutes les assignations. Une formule est fausse si elle n'est satisfaite pour aucune assignation.

La définition de la conséquence en logique du premier ordre fait intervenir non seulement les modèles, mais aussi les assignations. La conséquence standard, c'est-à-dire au sens conditionnel, se définit ainsi :

Définition 6.2. B est une conséquence de Λ ($\Lambda \models B$) si et seulement si dans tous les modèles, toutes les assignations $\mathcal{A}$ qui satisfont les formules membres de Λ satisfont également B. Ce qui se note :

$$A \models B \Leftrightarrow \forall \mathcal{M} \forall \mathcal{A} (\mathcal{M}, \mathcal{A} \models \Lambda \Rightarrow \mathcal{M}, \mathcal{A} \models B).$$

La conséquence logique au sens de la généralité se définit comme suit :

Définition 6.3. B est une conséquence de Λ au sens de la généralité ($\Lambda \models^x B$) si et seulement si dans tous les modèles, si toutes les assignations satisfont toutes les formules de Λ, alors toutes les assignations satisfont B. Ce qui formellement se note :

$$A \models^x B \Leftrightarrow \forall \mathcal{M} [\forall \mathcal{A} (\mathcal{M}, \mathcal{A} \models \Lambda) \Rightarrow \forall \mathcal{A} (\mathcal{M}, \mathcal{A} \models \Lambda \Rightarrow \mathcal{M}, \mathcal{A} \models B)].$$

6.4.3. *Force des deux relations de conséquence en logique du premier ordre*

La simple inspection de la position des quantificateurs dans la définition fait apparaître que la conséquence standard est une relation plus forte que la conséquence au sens de la généralité. En d'autres termes, on a :

$$\Lambda \models B \Rightarrow \Lambda \models^x B,$$

mais :

$$\Lambda \models^x B \not\Rightarrow \Lambda \models B.$$

L'exemple suivant, adapté de Kleene, permet de montrer que $\models$ est plus fort que $\models^x$.

Exemple. Soit un domaine comportant deux individus $D = \{1,2\}$ et un langage réduit à un prédicat P. Il y a donc deux assignations possibles à la variable x, et quatre interprétations possibles de P. P peut avoir comme extension l'ensemble vide, le singleton $\{1\}$, le singleton $\{2\}$ ou le domaine entier $\{1,2\}$. Représentons toutes les combinaisons possibles dans une table.

	P	x	$\neg Px$	$\forall x \neg Px$
a	$\{1,2\}$	1	**F**	**F**
b	$\{1,2\}$	2	**F**	**F**
c	$\{1\}$	1	**F**	**F**
d	$\{1\}$	2	**V**	**F**
e	$\{2\}$	1	**V**	**F**
f	$\{2\}$	2	**F**	**F**
g	$\emptyset$	1	**V**	**V**
h	$\emptyset$	2	**V**	**V**

Voyons à présent quelles sont les lignes qu'il faut inspecter pour vérifier respectivement $\neg Px \models \forall x \neg Px$ et $\neg Px \models^x \forall x \neg Px$.

Pour calculer $\neg Px \models \forall x \neg Px$ au sens standard, il faut passer en revue tous les cas où $\neg Px$ est satisfait et s'assurer que toute valeur assignée à x qui satisfait $\neg Px$, satisfait aussi $\forall x \neg Px$. On constate ainsi que ce n'est pas le cas. Pour l'assignation de 2 à x, la formule ouverte $\neg Px$ est satisfaite puisque $2 \notin \{1\}$. En revanche, la formule fermée $\forall x \neg Px$ n'est pas satisfaite pour cette même assignation. En effet, $\forall x \neg Px$ dit que rien n'est dans l'ensemble $\{1\}$. Elle est donc fausse, donc elle n'est satisfaite pour aucune assignation. En particulier, elle n'est pas satisfaite pour l'assignation de 2 à x qui satisfaisait $\neg Px$. Il y a même deux lignes (respectivement la ligne d et la ligne e) qui rendent vrai l'antécédent $\neg Px$ quand la variable x est entendue au sens conditionnel et qui rendent faux le conséquent $\forall x \neg Px$.

Passons maintenant au cas de la conséquence logique au sens de la généralité. Pour calculer $\neg Px \models^x \forall x \neg Px$ au sens de la généralité, il suffit de regarder le cas des lignes g et h car ce sont les seules où la formule $\neg Px$ comprise au sens de $\forall x \neg Px$ soit vraie, donc ce sont les seules lignes où l'affirmation d'une relation de conséquence risque d'être falsifiée. Certes, il s'agit ici d'une conséquence logique triviale puisque l'antécédent A et le conséquent B sont identiques. Mais en dépit de sa trivialité, l'exemple permet d'illustrer le point essentiel. Il nous montre que les conditions à remplir pour qu'un lien de *conséquence selon la généralité* existe entre A et B, sont un sous-ensemble des conditions à remplir pour qu'un lien de *conséquence au sens standard* existe entre A et B.

6.4.4. *La conséquence locale*

Dans la pratique, on fait des inférences à partir de propositions qui ne sont pas valides, c'est-à-dire vraies dans tous les mondes, mais qui sont seulement vraies factuellement ou localement, c'est-à-dire dans certains mondes. C'est pour rendre compte de cette sorte d'inférence que les théoriciens de la logique modale ont défini la relation de conséquence locale.

Définition 6.4. B est une conséquence locale de l'ensemble Λ de formules relativement à une classe de modèles $C(\mathcal{M})$ si et seulement si pour tous ces modèles $\mathcal{M}$ (c'est-à-dire pour toute structure $\mathcal{F}$, pour toute valuation V et pour tout monde w), si tous les membres de Λ sont vrais en w, B est vrai en w aussi. Cette définition a la forme:

(6.3) $\quad \Lambda \models_l B$ ssi $\forall \mathcal{M} \forall w \, (\mathcal{M} \models_w \Lambda \Rightarrow \mathcal{M} \models_w B)$.

6.4.5. *La conséquence globale pour les modèles*

Définition 6.5. B est une conséquence globale de l'ensemble Λ de formules relativement au modèle $\mathcal{M}$ si et seulement si, lorsque les formules de l'ensemble Λ sont vraies dans tous les mondes de $\mathcal{M}$, alors la formule B est vraie dans tous les mondes de $\mathcal{M}$. Cette définition a la forme :

$$(6.4) \qquad \Lambda \models_{g(\mathcal{M})} B \text{ ssi } \forall \mathcal{M} \in C(\mathcal{M})[\forall w\,(\mathcal{M} \models_w \Lambda) \Rightarrow \forall w\,(\mathcal{M} \models_w B)].$$

La règle de substitution n'est pas correcte quand on prend "conséquence logique" dans le sens de "conséquence logique globale". En effet, la règle de substitution ne transmet pas la validité au sens de vérité dans tous les mondes. On a décrit dans la figure 6.1 un modèle dans lequel $\Box p \supset p$, c'est-à-dire l'axiome T, est vrai dans tous les mondes du modèle, mais où le théorème $\Box q \supset q$ obtenu par substitution dans T n'est pas vrai dans tous les mondes du même modèle. En d'autres termes, il existe un modèle, c'est-à-dire une structure munie d'une valuation, qui rend vrai le membre de Λ où $\Lambda = \{\Box p \supset p\}$ et qui rend B faux, où $B = \Box q \supset q$.

En revanche, la règle de nécessité est correcte pour ce sens de "conséquence logique". Considérons un modèle dans lequel p est vrai dans chaque monde. Il est évident que p est *a fortiori* vrai dans chaque monde accessible. Cela montre que le lien qui unit la prémisse à la conclusion dans la règle de nécessité est une relation de conséquence globale définie en termes de modèle. On avait vu au paragraphe précédent que ce lien n'était pas la relation de conséquence locale.

6.4.6. *La conséquence globale pour les structures*

Définition 6.6. B est une conséquence globale de l'ensemble Λ de formules relativement à une classe de structures $C(\mathcal{F})$ si et seulement si, pour toute structure $\mathcal{F}$, lorsque toutes les formules membres de Λ sont valides dans $\mathcal{F}$, alors B est valide dans $\mathcal{F}$. Cette définition a la forme :

$$\Lambda \models_{g(\mathcal{F})} B \text{ ssi } \forall \mathcal{F} \in C(\mathcal{F})[\forall V \forall w(\mathcal{F}, V \models_w \Lambda) \Rightarrow \forall V, \forall w(\mathcal{F}, V \models_w B)].$$

Pour ce sens de "conséquence logique globale", les règles de substitution et d'échange sont toutes les deux correctes. Quant à la règle du *modus ponens*, elle est correcte pour les trois acceptions de la conséquence logique.

6.4.7. *Force des trois relations de conséquence en logique modale*

Récapitulons la présentation formelle des trois relations de conséquence logique :

$$\forall \mathcal{M} \in C(\mathcal{M}), \forall w\,(\mathcal{M} \models_w \Lambda \Rightarrow \mathcal{M} \models_w B)\,;$$
$$\forall \mathcal{M} \in C(\mathcal{M}), (\forall w\,(\mathcal{M} \models_w \Lambda) \Rightarrow \forall w\,(\mathcal{M} \models_w B))\,;$$
$$\forall \mathcal{F} \in C(\mathcal{F}), (\forall V \forall w\,(\mathcal{F}, V \models_w \Lambda) \Rightarrow \forall V \forall w\,(\mathcal{F}, V \models_w B))\,.$$

Dans la suite, nous travaillerons uniquement avec la première et la troisième de ces deux relations de conséquence logique : *la conséquence logique locale qui transmet la vérité* et *la conséquence globale qui transmet la validité* dans tous les modèles d'une structure ou d'une classe de structures. Nous avons :

$$\Lambda \models_l B \Rightarrow \Lambda \models_{g(\mathcal{F})} B.$$

Les réciproques ne sont pas vraies.

L'implication $(\Lambda \models_l B \Rightarrow \Lambda \models_g B)$ repose sur la loi qui autorise de distribuer le quantificateur universel sur l'antécédent et le conséquent d'un conditionnel :

$$\forall \mathcal{M}, \forall w\,(\mathcal{M} \models_w \Lambda \Rightarrow \mathcal{M} \models_w B) \Rightarrow \forall \mathcal{M}\,[\forall w\,(\mathcal{M} \models_w \Lambda) \Rightarrow \forall w\,(\mathcal{M} \models_w A)]\,.$$

La réciproque de cette première implication $(\Lambda \models B \Rightarrow \Lambda \models B)$ est fausse. Elle a la forme :

$$\forall \mathcal{M}\,[\forall w P(\mathcal{M}, w) \Rightarrow \forall w\, Q(\mathcal{M}, w)] \Rightarrow \forall \mathcal{M}, \forall w\ (P(\mathcal{M}, w) \Rightarrow Q(\mathcal{M}, w))\,,$$

qui n'est pas une vérité de logique. Un raisonnement analogue explique la différence entre $g(\mathcal{M})$ et $g(\mathcal{F})$, mais il s'agit maintenant de quantification sur les valuations V et non plus sur les mondes w.

6.4.8. *Importance de la conséquence locale*

Une première raison de travailler avec les deux notions de conséquence logique (la conséquence locale et la conséquence globale) est qu'il y a des principes de la logique classique, tels que 6.3 et 6.4, ci-dessous, qui sont conservés en logique modale quand on utilise la conséquence locale, mais qui sont perdus quand on utilise la conséquence globale.

En logique classique, la relation de conséquence logique obéit notamment aux deux principes suivants :

Principe 6.3. $\Lambda \cup \{A\} \models B$ si et seulement si $\Lambda \models A \supset B$ (analogue sémantique du théorème de la déduction).

Principe 6.4. $\Lambda \cup \{A\} \models C$ et $\Lambda \cup \{B\} \models C$ si et seulement si $\Lambda \cup \{A \vee B\} \models C$.

Ces deux principes restent corrects en logique modale dans les directions $\Rightarrow$ et $\Leftarrow$ quand on prend $\models$ au sens de la conséquence locale. Ils sont incorrects dans la direction $\Rightarrow$ quand on prend $\models$ au sens de la conséquence globale.

Considérons une instance de 6.3 où Λ est vide, A reste A et $\Box A$ est pris pour B. De $A \models \Box A$, on ne peut pas tirer la conséquence (au sens global du terme) que $\models_g A \supset \Box A$. Pour démontrer la non-validité de cette inférence, nous construirons un contre-exemple, c'est-à-dire un modèle $\mathcal{M}$ dans lequel il est vrai que si tous les mondes w rendent vrai A, alors tous les mondes w rendent vrai $\Box A$, mais dans lequel il est faux que tous les mondes w rendent vraie la proposition $A \supset \Box A$. Considérons le modèle $\mathcal{M}$ dont la structure $\mathcal{F}$

est constituée de $W = \{w_0, w_1\}$ et de $R = \{<w_0, w_1>\}$ et dont la valuation est : $V(A, w_0) = \mathbf{V}$, $V(A, w_1) = \mathbf{F}$ (voir figure 6.3).

w_1 • $\neg A$

↑

w_0 • A

Figure 6.3

Dans ce modèle, si $\forall w\ V(A, w) = \mathbf{V}$, alors $\forall w\ V(\Box A, w) = \mathbf{V}$ trivialement en vertu de la fausseté de l'antécédent, mais il n'est pas vrai pour tout w que $V(A \supset \Box A) = \mathbf{V}$. En effet, dans ce modèle il existe un monde, à savoir w_0 dans lequel A est vrai et $\Box A$ est faux.

Montrons à présent que de $A \models_g C$ et de $B \models_g C$, on ne peut tirer comme conséquence que $A \vee B \models_g C$ lorsque la conséquence en cause est la conséquence globale. Ici encore, on va construire un contre-exemple. On cherche un modèle dans lequel, si tous les mondes rendent vrai A, alors tous les mondes rendent vrai C, si tous les mondes rendent vrai B, alors tous les mondes rendent vrai C, mais où il n'est pas vrai que si tous les mondes rendent vrai $A \vee B$, alors tous les mondes rendent vrai C.

Le modèle $\mathcal{M}$ recherché se présente comme suit. On a deux mondes w_0 et w_1 non connectés. En d'autres termes, la relation d'accessibilité de $\mathcal{M}$ est la relation vide. Dans l'exemple de la figure 6.4, $A \models_g C$ et $B \models_g C$ sont vrais trivialement (parce que A n'est pas vrai dans tous les mondes et que, de même, B n'est pas vrai dans tous les mondes). De plus, tous les mondes (w_0 et w_1) rendent vrai $A \vee B$. En effet, w_0 rend vrai A, il rend donc également vrai $A \vee B$. D'autre part, w_1 rend vrai B, il rend donc également vrai $A \vee B$. Donc *tous* les mondes rendent finalement vraie la disjonction $A \vee B$. Mais tous les mondes ne rendent pas vrai C. En particulier, w_0 ne rend pas vrai C. Donc $A \vee B$ n'a pas pour conséquence globale C. Nous avons donc obtenu un contre-exemple à $A \models_g C$ et $B \models_g C \Rightarrow A \vee B \models_g C$.

w_1 • $\neg A, B, C$

w_0 • $A, \neg B, \neg C$

Figure 6.4

Ce raisonnement se représente formellement de la façon suivante :

$$A \models_g C \quad \Leftrightarrow \quad \underbrace{\left[\underbrace{\forall w(w \models A)}_{\boldsymbol{F}} \Rightarrow \forall w(w \models C)\right]}_{\boldsymbol{V}}$$

$$B \models_g C \quad \Leftrightarrow \quad \underbrace{\left[\underbrace{\forall w(w \models B)}_{\boldsymbol{F}} \Rightarrow \forall w(w \models C)\right]}_{\boldsymbol{V}}$$

$$A \vee B \models_g C \quad \Leftrightarrow \quad \underbrace{\left[\underbrace{\forall w(w \models A \vee B)}_{\boldsymbol{V}} \Rightarrow \underbrace{\forall w(w \models C)}_{\boldsymbol{F}}\right]}_{\boldsymbol{F}}$$

Le modèle qui vient d'être présenté n'aurait pas été un contre-exemple pour la conséquence locale. En effet, pour avoir un contre-exemple, il faudrait que le modèle vérifie les propositions suivantes :

1. Tous les mondes qui rendent vrai A rendent vrai C.
2. Tous les mondes qui rendent vrai B rendent vrai C.

Il faudrait en plus que le modèle rende fausse la proposition suivante :

3. Tous les mondes qui rendent vrai $A \vee B$ rendent vrai C.

Le modèle présenté dans la figure 6.4 ne remplit pas cette condition. En effet, w_0 rend vrai A et faux C. Donc le modèle rend (1) faux. Il ne peut donc rendre vrais (1) et (2) et faux (3).

Formellement :

$$A \models_l C \quad \Leftrightarrow \quad \underbrace{[\forall w(w \models A \Rightarrow w \models C)]}_{\boldsymbol{F}}$$

donc :

$$\underbrace{\underbrace{A \models_l C}_{\boldsymbol{F}} \text{ et } B \models_l C \Rightarrow A \vee B \models_l C}_{\boldsymbol{V}}$$

6.4.9. *Conséquence globale sur les structures et règle d'échange*

En logique classique, on a des principes d'échange parmi lesquels le suivant :

Principe 6.5. Si p est matériellement équivalent à q. c'est-à-dire s'il a la même valeur de vérité que q, et si C_q diffère de C_p en cela qu'on a remplacé une ou plusieurs occurrences de "p" par "q", alors C_q est matériellement équivalent à C_p.

Il est évident que ce principe, transféré simplement à la logique modale, ferait s'effondrer les distinctions modales (voir l'argument du lance-pierres, sect. 9.8). Il autoriserait en effet le raisonnement suivant :

$$\begin{aligned} & \Box(p \supset q), \\ & q \equiv r, \\ \text{donc} \quad & \Box(p \supset r). \end{aligned}$$

Ce résultat non désiré ne se produit pas si l'on prend "B équivalent à A" dans le sens "B est une conséquence de A et A est une conséquence de B *au sens de la conséquence globale sur les structures*". Moyennant cette précision, on peut formuler le principe suivant :

Principe 6.6. Si C_A et C_B sont semblables à cela près que C_B contient B à certaines places où C_A contient A et si A est équivalent à B au sens de la *conséquence globale*, alors C_A est équivalent à C_B.

Dans l'exemple ci-dessus, le remplacement de q par r n'est autorisé que si l'équivalence $q \equiv r$ tombe dans la portée de l'opérateur $\Box$.

Dorénavant, nous utiliserons uniquement les deux notions suivantes de conséquence :

1. La conséquence locale.
2. La conséquence globale sur les structures.

6.4.10. *Le lien unissant la conséquence locale à la conséquence globale*

Au paragraphe 6.4.7, on a démontré un résultat positif :

$$\Lambda \models_l B \Rightarrow \Lambda \models_g B,$$

et un résultat négatif :

$$\Lambda \models_g B \not\Rightarrow \Lambda \models_l B.$$

La règle de nécessité par laquelle on infère $\Box B$ de B est correcte si B est valide, mais elle ne l'est pas si B est simplement vrai. Elle reflète donc un lien de conséquence globale.

Si on ne considère que la classe des modèles fermés sur l'opération qui, à partir de $\mathcal{M}$, produit un sous-ensemble $\mathcal{M}'$ qui soit *sous-ensemble engendré* de $\mathcal{M}$ (voir définitions au § 7.3.2), l'implication vaut dans les deux directions, à condition toutefois que l'on renforce les prémisses locales à l'aide de $\Box$.

On appelle clôture de l'ensemble Λ de formules par l'opérateur $\Box$, l'opération qui consiste à préfixer les formules membres de cet ensemble par des suites finies d'opérateurs de nécessité.

On définit récursivement cette opération de clôture ainsi :

- $\Lambda_0 = \Lambda$;
- $\Lambda_{n+1} = \Lambda_n \cup \{\Box_i A : A \in \Lambda_n \text{ et } i \in I\}$;
- $\Box\Lambda = \bigcup_{n \in w} \Lambda_n$.

Concrètement, il s'agit d'ajouter un opérateur de nécessité aux prémisses locales autant de fois que l'on a appliqué la règle de nécessité aux prémisses globales.

Moyennant le renforcement des prémisses locales par le nombre approprié d'opérateurs $\Box$, on peut relier la *conséquence locale* $\models_l$ à la *conséquence globale* $\models_g$ au moyen du théorème suivant [Jansana 95].

Théorème 6.7. Soit $C(\mathcal{M})$ une classe de modèles fermée sur *la formation de sous-modèles engendrés.* Alors, pour toute classe Λ de formules modales et pour n'importe quelle formule B, l'ensemble Λ a pour conséquence globale B si et seulement si cette même classe Λ fermée pour la nécessité a pour conséquence locale B, ce qui formellement se note :

$$\Lambda \models_g B \Leftrightarrow \Box\Lambda \models_l B.$$

Relativement à la classe de modèles précitée, la conséquence globale est équivalente à la conséquence locale renforcée.

6.5. Le théorème de la déduction en logique modale

6.5.1. *Le théorème de la déduction en logique modale et en logique non modale*

Sous sa forme syntaxique, le théorème de la déduction s'énonce ainsi :

Théorème 6.8. Si B est dérivable de l'ensemble de prémisses Λ et de la prémisse A, alors $A \supset B$ est dérivable de Λ, ce qui formellement se note :

$$\frac{\Lambda, A \vdash B}{\Lambda \vdash A \supset B}.$$

Le calcul des prédicats comporte une règle, la règle de généralisation universelle (*G.U.*), qui permet de dériver $\forall x\, Fx$ de Fx. Si l'on appliquait successivement le théorème de la déduction puis la règle de généralisation universelle, on démontrerait des théorèmes faux.

Exemple. Par exemple, on démontrerait l'induction ordinaire :

1. $Fx \vdash \forall x\, Fx$;
2. $\vdash Fx \supset \forall x\, Fx$, (théorème de la déduction);
3. $\vdash \forall x(Fx \supset \forall x\, Fx)$, (*G.U.*);
4. $\vdash \exists x\, Fx \supset \forall x\, Fx$, (loi de passage).

Pour empêcher les dérivations de ce genre, le théorème de la déduction en logique du premier ordre doit être assorti de restrictions. On le formule ainsi :

Théorème 6.9. Supposons qu'il existe une dérivation de B à partir des formules contenues dans Λ, A. Si aucune application de la règle de généralisation universelle ne quantifie universellement sur une variable libre présente dans une formule qui dépend de A, alors $\Lambda \vdash A \supset B$ [Mendelson 64], p. 61.

En logique modale, la règle de nécessité s'apparente à la règle de généralisation universelle en logique du premier ordre. Si l'on traduit la logique modale propositionnelle en logique du premier ordre – ce que l'on peut faire pour les systèmes les plus courants au moyen d'un algorithme de traduction –, la règle de nécessité devient un cas particulier de la règle de généralisation universelle, à savoir une règle qui, au lieu de quantifier sur une variable dont le domaine est universel, quantifie sur des individus bien particuliers : les mondes possibles bornés par la relation d'accessibilité.

Après traduction dans le premier ordre, la règle de nécessité

$$\frac{A}{\Box A},$$

s'écrit :

$$\frac{A(w)}{\forall w A(w)}.$$

Au niveau de la théorie de la preuve, on a vu que la *règle de généralisation universelle* et le *théorème de la déduction* s'excluaient mutuellement, sauf si des restrictions appropriées étaient introduites (comme en 6.9). On verra de même que la *règle de nécessité* et le *théorème de la déduction* s'excluent mutuellement. En simplifiant, on peut dire que la règle est applicable quand le théorème ne l'est pas et réciproquement.

La théorie de la preuve en logique modale se complique du fait que l'on a reconnu deux sortes de conséquences : la conséquence locale et la conséquence globale. Pour refléter cette distinction sémantique, dans la théorie de la preuve, Jansana distingue deux relations de déductibilité : la relation de *déductibilité faible* et la *relation de déductibilité forte* [Jansana 95]. Fitting quant à lui, distingue un *théorème de la déduction local* et un *théorème de la déduction global* [Fitting 93]. Cette dernière distinction s'appuie sur une distinction plus fondamentale qui relève aussi de la théorie de la preuve : la distinction entre *prémisses locales* et *prémisses globales.* Cette dernière distinction, bien que syntaxique, a une justification sémantique qui en éclaire la portée.

6.5.2. *Les deux sortes de prémisses*

Tant qu'on se cantonne dans les systèmes axiomatiques qui ne font pas de place à des prémisses distinctes des axiomes, la logique modale est une extension conservatrice de la logique non modale, préservant tous ses axiomes et théorèmes et ajoutant de nouveaux axiomes (K, T, 4, B, C, D etc...) ainsi qu'une nouvelle règle, la règle de nécessité, qui, avec les axiomes, permet la déduction de théorèmes nouveaux.

Si on vise à utiliser la méthode axiomatique (ou d'autres techniques de preuve) pour des applications extérieures à la logique, on doit redéfinir la notion de preuve et préciser les conditions d'emploi des hypothèses extérieures à la logique I, p. 126-131. Déjà en logique propositionnelle non modale, il y a des restrictions à la règle de substitution uniforme. La règle s'applique aux axiomes, mais pas aux hypothèses.

La logique modale, à la différence de la logique classique, distingue les prémisses constituées par des propositions tenues provisoirement pour des vérités de logique, des prémisses tenues provisoirement pour des vérités factuelles. Si les déductions à partir de prémisses sont plus compliquées en logique modale qu'en logique classique, c'est parce que, au sein des prémisses extra-logiques, nous devons fournir une machinerie pour les prémisses qui sont supposées être factuellement vraies ainsi que pour les prémisses qui sont supposées être logiquement vraies. La distinction entre les deux se réfléchit dans la manière dont la règle de nécessité peut être appliquée [Fitting 93b].

Fitting a introduit les dénominations de "suppositions globales" et de "suppositions locales". Il divise l'ensemble des prémisses (distinctes des axiomes) en (1) suppositions globales qui valent dans tous les mondes possibles et (2) suppositions locales qui valent seulement dans certains mondes. Pour refléter cette distinction dans le formalisme, il redéfinit la relation syntaxique de dérivabilité en ces termes :

(6.5) $\quad S \models U \rightarrow X,$

où S est l'ensemble des prémisses globales et U l'ensemble des prémisses locales.

6.5.3. *Systèmes non normaux*

Même dans un système axiomatique (sans prémisse), la règle de nécessité ne s'applique pas toujours. On a alors affaire à un système *non normal*.

C'est le cas du système S_2 de Lewis. Ce système non normal comporte les axiomes suivants :

1. Toute tautologie du calcul des propositions est un axiome.
2. $\Box p \supset p$.
3. $(\Box (p \supset q) \wedge \Box (q \supset r)) \supset \Box (p \supset r)$.
4. $\Box (\Diamond (p \supset q) \wedge (\Diamond p \supset \Diamond q))$.

C'est aussi le cas du système que l'on obtient en ajoutant aux axiomes de S_4 l'axiome de Mc Kinsey à savoir $\Box\Diamond p \supset \Diamond \Box p$.

6.5.4. *Le théorème de la déduction revisité*

Le théorème classique de la déduction dans sa version sémantique s'exprime au moyen des deux schémas suivants :

$$\text{(6.6)} \qquad \frac{\Gamma, A \models B}{\Gamma \models A \supset B},$$

$$\text{(6.7)} \qquad \frac{\Gamma \models A \supset B}{\Gamma, A \models B}.$$

Si l'on veut faire place à la distinction entre prémisses locales et prémisses globales, on a besoin d'un formalisme plus élaboré.

Soit $C(\mathcal{F})$ une collection de structures. Soit S un ensemble de suppositions globales, c'est-à-dire un ensemble de formules vraies dans tous les mondes possibles. Soit U un ensemble de prémisses locales dont on exige seulement qu'elles soient vraies dans certains mondes particuliers. Fitting propose de reformuler ainsi la ligne-prémisse du théorème de la déduction :

$$S \models_{C(\mathcal{F})} U \to X,$$

ce qui se lit "Pour toute structure $\mathcal{F} = < W, R >$ de la classe de structures $C(\mathcal{F})$ et pour tout modèle $\mathcal{M} = < W, R, V >$ basé sur $\mathcal{F}$ dans lequel les formules de l'ensemble S sont valides, tous les mondes possibles qui rendent vraies les formules de U, rendent vraies aussi les formules de X".

On notera que la flèche $(\to)$ représente une relation de conséquence locale et que le tourniquet $(\models)$ représente un lien de conséquence globale.

Exactement comme les axiomes des systèmes normaux, les prémisses globales autorisent l'application de la règle de nécessité, contrairement aux prémisses locales.

6.5.5. *Le théorème local de la déduction*

Sous sa forme locale, le théorème de la déduction ne pose aucun problème. Il se formule ainsi pour n'importe quelle classe de structures $C(\mathcal{F})$:

$$\text{(6.8)} \qquad \frac{S \models_{C(\mathcal{F})} U \cup \{X\} \to Y}{S \models_{C(\mathcal{F})} U \to (X \supset Y)}.$$

Observons que la prémisse locale X passe du statut de prémisse à celui d'antécédent d'un conditionnel. Dans (6.9), qui est la réciproque de (6.8), c'est l'inverse :

$$\text{(6.9)} \qquad \frac{S \models_{C(\mathcal{F})} U \to (X \supset Y)}{S \models_{C(\mathcal{F})} U \cup \{X\} \to Y}.$$

La justification du théorème local de la déduction repose sur le fait que le connecteur du conditionnel ($\supset$) se comporte classiquement, quel que soit le monde possible dans lequel on l'évalue.

6.5.6. *Le théorème global de la déduction*

Le théorème global de la déduction correspondant au schéma (6.6) est restreint par la condition suivante : il s'applique uniquement aux classes de structures $C(\mathcal{F}')$ fermées sur les *sous-structures engendrées* et sur les *ultraproduits*. Il s'énonce ainsi :

$$(6.10) \qquad \frac{S \cup \{X\} \models_{C(\mathcal{F}')} U \to Y}{S \models_{C(\mathcal{F}')} U \cup \{\Box^0 X, \Box^1 X, \ldots, \Box^n X\} \to Y}.$$

On observe que la prémisse X qui faisait partie de la prémisse *globale* ($S \cup \{X\}$) dans la première ligne du schéma devient l'antécédent *local* d'un conditionnel dans la seconde ligne.

Cette modification de statut (le passage de "globale" à "locale") entraîne un affaiblissement puisque, comme on l'a rappelé plus haut, la règle de nécessité s'applique aux prémisses globales, mais non aux prémisses locales.

Pour annuler l'effet du passage du global au local, on renforce l'antécédent X en le préfixant d'autant d'opérateurs de nécessité qu'on a utilisé de fois la règle de nécessité dans la dérivation.

Fitting décrit ainsi ce *mécanisme de compensation* : *Si une supposition globale est utilisée dans une dérivation, la règle de nécessité peut lui être appliquée.*

Comme les dérivations sont finies, la règle de nécessité ne peut être appliquée qu'un nombre fini de fois. Dès lors, utiliser X comme supposition globale doit revenir à utiliser $\Box\Box \ldots \Box X$ comme supposition locale pour les différentes séquences de symboles [Fitting 93], p. 381-382.

6.6. La preuve de complétude par la méthode du modèle canonique

6.6.1. *Correction, complétude, caractérisation pour les systèmes sans prémisses*

Les notions suivantes ont été introduites dans la section 6.2 :

1. *Correction* : $\Lambda \vdash B \Rightarrow \Lambda \models B$.
2. *Complétude* : $\Lambda \models B \Rightarrow \Lambda \vdash B$.
3. *Adéquation* : $\Lambda \vdash B \Leftrightarrow \Lambda \models B$.

Quand la classe des prémisses (Λ) est vide, comme c'est le cas des systèmes axiomatiques présentés dans les chapitres 2 et 3, les définitions prennent la forme suivante :

1. *Correction* : tout théorème est valide, ce qui se note : $\vdash B \Rightarrow \models B$.
2. *Complétude* : toute formule valide est un théorème, soit : $\models B \Rightarrow \vdash B$.
3. *Adéquation* : $\vdash B \Leftrightarrow \models B$.

Lorsque Λ est fini, on peut prouver facilement, par applications successives du théorème de la déduction :

$$A_1 \ldots A_n \vdash B \Leftrightarrow \vdash A_1 \rightarrow (A_2 \rightarrow (\cdots \rightarrow B) \ldots).$$

Ensuite par application du théorème d'importation, on obtient :

$$A_1 \ldots A_n \vdash B \Leftrightarrow \vdash A_1 \wedge \cdots \wedge A_n \rightarrow B.$$

Lors des applications successives du théorème de la déduction, il était entendu que la dérivation $\Lambda \vdash A$ n'avait jamais fait usage de la règle de nécessité.

Dans la preuve de complétude présentée, on fera uniquement appel au théorème de la déduction faible (c'est-à-dire comportant un nombre fini de preuves).

6.6.2. *Caractérisation*

Un système de logique L caractérise une classe $C(\mathcal{F})$ de structures si et seulement s'il est correct et complet par rapport à cette classe. Les systèmes de logique modale ne sont pas destinés uniquement à contrôler la rigueur des raisonnements dans lesquels les expressions "nécessairement", "toujours", "obligatoirement", "il est su que", jouent le rôle de mots logiques comme les connecteurs, les variables d'individus et les quantificateurs le font dans la logique du premier ordre. Ils remplissent aussi un autre rôle : celui de décrire des classes de structures (de modèles) : structures du temps, de l'obligation, de la connaissance. Il faut comprendre ici "décrire" dans le sens technique de "caractériser" qui a été défini plus haut.

La théorie des modèles pour la logique modale étudie systématiquement les relations de caractérisation qui existent entre systèmes formels et classes de structures. Un même système formel peut caractériser des classes différentes de structures. Rappelons qu'une structure $\mathcal{F} = <W, R>$ est la classe des modèles $\mathcal{M} = <W, R, V>$ où V est quelconque.

6.6.3. *La complétude prouvée à partir du modèle canonique*

Les preuves de complétude diffèrent selon le type de méthode de preuve choisi. Pour prouver la complétude des systèmes axiomatiques les plus courants, tels que les quinze systèmes décrits au chapitre 2, il existe une

méthode devenue standard. C'est la *méthode de preuve par modèle canonique.* Cette méthode de preuve est particulièrement robuste. En effet, elle est aisément transposable d'un système axiomatique à un autre, moyennant de petites modifications. On trouvera ci-dessous la preuve de complétude de K par cette méthode.

Pour prouver la complétude : toute formule valide dans un modèle de K est un théorème, c'est-à-dire :

(6.11) $\models_{C(\mathcal{M})} A \Rightarrow \vdash_L A,$

on prouve généralement la contraposée : tout non-théorème de L est non valide dans la classe $C(\mathcal{M})$, c'est-à-dire :

(6.12) $\nvdash_L A \Rightarrow \not\models_{C(M)} A.$

La proposition (6.12) peut à son tour être reformulée ainsi :
Pour tout non-théorème A de L, il existe un modèle membre de $C(\mathcal{M})$ dans lequel la formule A est fausse, ce qui se note :

(6.13) $\forall A$ tel que $\nvdash_L A, \exists \mathcal{M}$ tel que $V(A, \mathcal{M}) = \mathbf{F}$.

La preuve de complétude par le modèle canonique vise à établir un résultat plus fort que la proposition (6.13), un résultat dont cette proposition découle. Ce résultat plus fort s'énonce :

Il existe un modèle dans lequel tous les non-théorèmes de K sont faux et tous les théorèmes de K sont vrais, ce qui se note :

(6.14) $\exists \mathcal{M}$ tel que $\forall A, \vdash_L A \Leftrightarrow V(A, \mathcal{M}) = \mathbf{F}$.

La proposition (6.14) est plus forte que la proposition (6.13). Au lieu d'avoir comme quantificateurs de tête $\forall A\ \exists \mathcal{M}$, la proposition (6.14) commence par $\exists \mathcal{M}\, \forall A$. La proposition (6.14) exige donc que le *même modèle* rende faux tous les non-théorèmes de S (et vrais tous les théorèmes de S), ce qui n'était pas le cas de la proposition (6.13).

6.6.4. *La structure canonique*

Définition 6.7. Un ensemble de formules Λ est *consistant maximal* si l'adjonction à cet ensemble d'une formule qui n'était pas déjà membre de cet ensemble, le rend inconsistant.

Comme tout modèle, le modèle canonique est constitué d'une structure $\mathcal{F} = <W, R>$ et d'une valuation V.

Dans une structure canonique, les *mondes* w membres de W sont des *ensembles de formules*, plus précisément des ensembles maximaux consistants de formules au sens de "maximal consistant" précisé plus haut, sens familier depuis la preuve donnée par *Henkin* en 1949 de la complétude de la logique du premier ordre, preuve différente de celle que *Gödel* publia en 1930.

La relation d'accessibilité R pour une structure canonique sera dès lors une

relation entre ensembles maximaux consistants de formules : sRt ssi $A \in s \Rightarrow A \in t$.

6.6.5. *La valuation canonique*

La valuation canonique assigne à chaque atome propositionnel une valeur relativement à un monde possible, c'est-à-dire relativement à un ensemble consistant maximal de formules selon la règle suivante :

Pour tout atome propositionnel p, la valeur de p évaluée dans le monde s, est le vrai (respectivement le faux) selon que p est membre (respectivement n'est pas membre) de s, ce qui se note :

$$V(p, s) = \mathbf{V} \Leftrightarrow p \in s,$$

ou, en utilisant la relation $\models$ (rend vrai) :

$$\models_s p \Leftrightarrow p \in s.$$

6.6.6. *Le modèle canonique*

Pour construire un *modèle* $\mathcal{M}$ à partir d'une *structure* $\mathcal{F} =< W, R >$ et d'une *valuation* V, il ne suffit pas d'assigner une valeur de vérité aux *atomes propositionnels* membres des ensembles maximaux consistants. Il faut encore assigner de manière cohérente une valeur de vérité à toutes les *formules* figurant dans ces ensembles. A cette fin, on étend la valuation V en V'. Comme les ensembles consistants maximaux sont généralement infinis, cette extension fait appel à une induction.

6.6.7. *Le système canonique*

Définition 6.8. Un système formel L est appelé système canonique si la structure $\mathcal{F} =<W, R>$ du modèle canonique $\mathcal{M} =<W, R, V>$ est une structure pour L, ce qui n'est pas toujours le cas.

Tout système canonique est complet. En effet, si le système L est canonique, il est, en vertu de la définition de la canonicité, complet relativement à la *structure canonique* $\mathcal{F}$ qui fait partie de son *modèle canonique*. Prouver la canonicité d'un système est donc une des manières de prouver sa complétude.

La réciproque n'est pas vraie. Tous les systèmes complets ne sont pas canoniques. Le système G, conçu pour capter la notion de prouvabilité au moyen de l'opérateur modal $\Box$, est complet, mais n'est pas canonique. G s'obtient en ajoutant à K l'axiome suivant :

$$\Box(p \wedge (\Box p \supset p)) \supset \Box p.$$

Ce système est complet relativement à la classe des structures finies et partiellement ordonnées (dans lesquelles la relation d'accessibilité R est

transitive et irréflexive). Mais il n'est pas complet relativement à la structure $\mathcal{F} = <W, R>$ de son propre modèle canonique. En effet, cette dernière contient au moins un monde relié à lui-même par une boucle, ce qui rend non valide l'axiome caractéristique dans G.

6.6.8. *La preuve de complétude*

6.6.8.1. *Introduction*

La preuve de complétude exposée ici est due à S. Popkorn [Popkorn 94]. Seule la présentation diffère, toutes les étapes ont été explicitées et tous les lemmes ont été démontrés.

6.6.8.2. *Définitions préliminaires*

Définition 6.9. La formule A est syntaxiquement inconsistante relativement à la logique L si et seulement si sa négation est démontrable dans L, ce qui se note $\vdash_L \neg A$.

Définition 6.10. La formule A est syntaxiquement consistante relativement à L si et seulement si sa négation n'est pas démontrable dans L, ce qui se note $\nvdash_L \neg A$.

Définition 6.11. L'ensemble fini de formules $\{A_1, \ldots, A_n\}$ est syntaxiquement consistant relativement à L si et seulement si la négation de la conjonction $A_1 \wedge \cdots \wedge A_n$ n'est pas démontrable dans L.

Définition 6.12. Un ensemble s de formules est consistant maximal relativement à L si et seulement s'il est consistant relativement à L et si l'adjonction de toute nouvelle formule le rend inconsistant. Dans le cas traité ci-dessous, $L = K$.

6.6.8.3. *Propriétés de clôture des ensembles maximaux consistants*

Propriété 6.1. Il n'existe pas de A tel que $A \in s$ et $\neg A \in s$ (par définition).

Propriété 6.2. $A \in s$ ou $\neg A \in s$.

Démonstration. Supposons le contraire, c'est-à-dire $A \notin s$ et $\neg A \notin s$.

Si $A \notin s$, alors $s \cup \{A\}$ est une extension propre, et comme s est maximal, toute extension le rend inconsistant, donc il y a un sous-ensemble fini de s, à savoir $\{B_1, \ldots, B_n\}$ tel que $B_1 \wedge \cdots \wedge B_n \wedge A \vdash \mathbf{F}$.

Si $\neg A \notin s$, alors $s \cup \{\neg A\}$ est une extension propre, et comme s est maximal, son extension est inconsistante, donc il y a un sous-ensemble fini de s, à savoir $\{C_1, \ldots, C_n\}$ tel que $C_1 \wedge \cdots \wedge C_n \wedge \neg A \vdash \mathbf{F}$.

Comme $B \wedge A \supset C$ est équivalent à $A \supset (B \supset C)$, on a :
$\vdash A \supset [(B_1 \wedge \cdots \wedge B_n) \supset \mathbf{F}]$ et $\vdash \neg A \supset [(C_1 \wedge \cdots \wedge C_n) \supset \mathbf{F}]$.

Appelons σ la conjonction $B_1 \wedge \cdots \wedge B_n$ et τ la conjonction $C_1 \wedge \cdots \wedge C_m$. De $A \wedge \sigma \supset \mathbf{F}$ et $\neg A \wedge \tau \supset \mathbf{F}$ on tire aisément $\sigma \wedge \tau \supset \mathbf{F}$. En appliquant $\vdash A \vee \neg A$ à $A \supset (\sigma \supset \tau)$ et à $\neg A \supset (\tau \supset \sigma)$, on obtient $\sigma \supset \mathbf{F} \vee \tau \supset \mathbf{F}$ et de là, $\overline{\sigma} \vee \overline{\tau} \vee \mathbf{F}$ et enfin $\sigma \wedge \tau \supset \mathbf{F}$.

Propriété 6.3. L'ensemble s est fermé par rapport à l'implication.
Démonstration. Supposons le contraire, c'est-à-dire que $A_1 \wedge \cdots \wedge A_n \in s$, que $\vdash A_1 \wedge \cdots \wedge A_n \supset B$ et que $B \notin s$. Si $B \notin s$, alors $s \cup \{B\}$ est inconsistant et il existe une conjonction finie de membres de s tels que $\vdash C_1 \wedge \cdots \wedge C_m \wedge B \supset \mathbf{F}$. Donc en appliquant la tautologie $[(p \supset q) \wedge [(r \wedge q) \supset t]] \supset [(p \wedge r) \supset t]$, on obtient $A_1 \wedge \cdots \wedge A_n \wedge C_1 \wedge \cdots \wedge C_m \supset \mathbf{F}$.

6.6.8.4. *Equivalences remarquables*

Rappelons que s est un ensemble consistant maximal.

1.

(6.15) $\neg A \in s \Leftrightarrow A \notin s$.

Démonstration. Utiliser les propriétés 6.1 et 6.2 ci-dessus et la définition de $\Rightarrow$.

2.

(6.16) $A \wedge B \in s \Leftrightarrow A \in s$ et $B \in s$.

Démonstration.

$\vdash A \wedge B \quad \Rightarrow \quad A, B$ et fermeture sur l'implication.
$\vdash A, B \quad \Rightarrow \quad A \wedge B$ et fermeture sur l'implication.

3.

(6.17) $A \vee B \in s \Leftrightarrow A \in s$ ou $B \in s$.

Démonstration.

(a) $A \in s$ ou $B \in s$.
$\vdash A \supset A \vee B, \vdash B \supset A \vee B$;
$A \vee B \in s$ (fermeture sur l'implication).

(b) $A \vee B \in s$.
Supposons que $A \notin s$ et $B \notin s$; $\neg A \in s$ et $\neg B \in s$ par maximalité;
$\neg A \wedge \neg B \in s$ par l'équivalence (6.16);
$\neg (A \vee B) \in s$. Alors s est inconsistant, contrairement à l'hypothèse.
Rejetant la supposition, on obtient $A \in s$ ou $B \in s$.

4.

(6.18) $A \supset B \in s \Leftrightarrow A \notin s$ ou $B \in s$.

Démonstration.

(a) $A \notin s$ ou $\in s$.
$\neg A \in s$ par maximalité;
$\vdash \neg A \supset (A \supset B)$;
$A \supset B \in s$ par fermeture sur l'implication;
$\vdash B \supset (A \supset B)$;
$A \supset B \in s$ par fermeture sur l'implication.

(b) $A \supset B \in s$.
$\vdash (A \supset B) \supset (\neg A \vee B)$;
$\neg A \vee B \in s$;
$\neg A \in s$ ou $B \in s$;
$A \notin s$ ou $B \in s$ par l'équivalence (6.15).

6.6.8.5. *Lemme d'existence (Lemme de Lindenbaum)*

Le résultat fondamental sur lequel repose le théorème de complétude démontré ici est le lemme qui affirme que, pour tout ensemble Λ consistant (relativement au système considéré), il existe un ensemble s consistant maximal qui le contient.

Pour prouver le lemme d'existence, on arrange toutes les formules de la logique modale selon un ordre : $A_1, A_2, A_3 \ldots$ Cette énumération est réalisable puisque l'ensemble de ces formules est dénombrable.

On construit une suite d'ensembles $\Lambda_0, \Lambda_1, \Lambda_2 \ldots$ de la façon suivante :
Si Λ est un ensemble consistant relativement à K,

$$\begin{aligned} \Gamma_0 &= \Lambda; \\ \Gamma_{n+1} &= \Gamma_n \cup \{A_{n+1}\} \text{ si l'ensemble est consistant par rapport à } K; \\ &= \Gamma_n \cup \{\neg A_{n+1}\} \text{ dans le cas contraire.} \end{aligned}$$

L'ensemble maximal s est obtenu en prenant l'union de tous les Γ_n. Il reste à prouver que s est consistant et maximal. Il est consistant parce que s'il ne l'était pas, l'un des Γ_i serait inconsistant, ce qui ne peut être le cas en raison de la construction. Pour la maximalité, voir [Popkorn 94], p. 122, 123.

6.6.8.6. *Le modèle canonique*

(a) W est l'ensemble des ensembles de formules qui sont K-consistants, c'est-à-dire consistants relativement à K, et maximaux.

(b) R est la relation entre ensembles K-consistants et maximaux telle que sRt si et seulement si pour tout A, $\Box A \in s \Rightarrow A \in t$.

(c) V est la valuation des atomes propositionnels telle que $V(p, s) = \mathbf{V}$ si et seulement si $p \in s$.

On étend ensuite cette valuation V en une valuation V' telle que la formule A satisfait $V'(A, s) = \mathbf{V}$ si et seulement si $A \in s$. Cette extension se fait au moyen d'une induction sur la complexité de A. Cette induction comporte une clause pour chacun des trois symboles primitifs de K : le connecteur de la négation, le connecteur du conditionnel et l'opérateur de nécessité. Les deux premières clauses sont standard (depuis Henkin 1949). La troisième clause est propre à la logique modale. Avant de prouver l'existence de V', deux lemmes préparatoires sont nécessaires : le lemme de dérivabilité et le lemme spécifiant les conditions que doit remplir une formule $\Box A$ pour appartenir à un ensemble consistant maximal.

6.6.8.7. *Lemme de dérivabilité*

Pour tout ensemble de formules Λ et toute formule B,

$$\Lambda \vdash B \Leftrightarrow \text{ pour tout } s \text{ membre de } S,\ \Lambda \subseteq s \Rightarrow B \in s;$$

(a)

$$\Lambda \vdash B \Rightarrow \text{ pour tout } s \text{ membre de } S,\ \Lambda \subseteq s \Rightarrow B \in s.$$

Démonstration. Appelons déductibilité faible la dérivabilité à partir d'un nombre fini de prémisses que l'on note $\vdash_f$.

Si $\Lambda \vdash_f B$, alors $\vdash (\forall A \in \Lambda)[A \supset B]$ en vertu du théorème de déduction pour la relation de déductibilité faible, qui est seule en cause ici, et en raison de la fermeture des s sur l'implication, on a $A_1 \ldots A_n \cdots \in s \Rightarrow B \in s$.

(b) Pour tout s membre de S, si $\Lambda \subseteq s \Rightarrow B \in s$, alors $\Lambda \vdash B$.

Démonstration. Par hypothèse, pour tout s, $\Lambda \subseteq s \Rightarrow B \in s$. Donc il n'y a pas de s tel que $\Lambda \subseteq s$ et $B \notin s$, donc pas de s tel que $\Lambda \subseteq s$ et $\neg B \in s$, c'est-à-dire pas de s tel que $\Lambda \subseteq s$ et $\{\neg B\} \subseteq s$. Or, en vertu du lemme d'existence, pour tout Λ, il existe un s tel que $\Lambda \subseteq s$.

Comme Λ est un ensemble et que $\{\neg B\}$ est un ensemble, $\Lambda \cup \{\neg B\}$ est un ensemble, mais un ensemble inconsistant. Il y a donc un sous-ensemble fini de Λ tel que $\vdash_L A_1 \wedge \cdots \wedge A_n \wedge \neg B \supset \mathbf{F}$. Donc $(A_1 \wedge \cdots \wedge A_n) \supset (\neg B \supset \mathbf{F})$. Donc $(A_1 \wedge \cdots \wedge A_n) \supset (\mathbf{V} \supset B)$ (par contraposition) et $A_1 \wedge \cdots \wedge A_n \vdash B$ (par simplification de $\mathbf{V} \supset B$ en B). D'où on tire, par monotonie, $\Lambda \vdash B$.

6.6.8.8. *Lemme préparatoire à l'extension de la valuation des atomes aux formules*

Pour tout s membre de **S** (l'ensemble des ensembles consistants maximaux),

$$\Box A \in s \equiv \forall t(sRt \supset A \in t);$$

(a) $\Box A \in s \supset \forall t(sRt \supset A \in t)$.

Démonstration. Cette formule s'obtient par permutation des antécédents dans $sRt \supset (\Box A \in s \supset A \in t)$ qui se tire de la définition de R pour un modèle canonique.

(b) $\forall t(sRt \supset A \in t) \supset \Box A \in s$.

Démonstration. La preuve de (b) est la partie la plus difficile du théorème de complétude. On se donne un ensemble consistant maximal s contenant des formules de la forme $\Box B$. On forme un nouvel ensemble composé de ces formules dont on a retiré l'opérateur de nécessité figurant comme opérateur principal. Pour l'ensemble $\Gamma = \{B|\Box B \in s\}$ ainsi formé, il existe (au moins) un ensemble consistant maximal t tel que $\Gamma \subseteq t$ en vertu du lemme d'existence. Ce qui se note :

(6.19) $\{B|\Box B \in s\} \subseteq t$.

Si on exprime au moyen du symbole d'appartenance, ce qui était exprimé par le symbole d'inclusion, on obtient :

(6.20) $\Box B \in s \supset B \in t$.

La formule (6.20) est la redéfinition de sRt. On remplace (6.20) par (6.21) :

(6.21) sRt.

On prend comme hypothèse :

(6.22) $\forall t(sRt \rightarrow A \in t)$,

c'est-à-dire l'antécédent de la formule à démontrer.

A partir de (6.21) et de (6.22), par instantiation universelle et *modus ponens*, on obtient :

(6.23) $A \in t$.

On a donc prouvé que (6.19) implique (6.23), c'est-à-dire :

(6.24) $\{B|\Box B \in s\} \subseteq t \Rightarrow A \in t$.

Par le lemme de dérivabilité, on obtient :

(6.25) $\{B|\Box B \in s\} \vdash A$.

Donc :

(6.26) $\vdash B_1 \wedge \cdots \wedge B_n \supset A$,

(compacité et théorème de déduction pour la déductibilité faible).

On applique la règle de nécessité à (6.26) qui est un théorème, puis la distribution de $\Box$ sur la conjonction et l'on obtient :

(6.27) $\vdash \Box B_1 \wedge \cdots \wedge \Box B_n \supset \Box A$.

Comme $\Box B_i \in s$ et comme les ensembles consistants maximaux sont fermés sur l'implication,

(6.28) $\Box A \in s$,

et, en déchargeant l'hypothèse (6.22), on obtient :

(6.29) $\forall t(sRt \supset A \in t) \supset \Box A \in s$.

6.6.8.9. *Le lemme d'extension de la valuation V en V'*

1. Clause pour la négation :

   (6.30) $V(\neg B, s) = \mathbf{V} \Leftrightarrow \neg B \in s$.

   Démonstration.

$$\begin{aligned} V(\neg B, s) = \mathbf{V} &\Leftrightarrow V(B, s) = \mathbf{F}; \\ V(B, s) = \mathbf{F} &\Leftrightarrow B \notin s \ (H.I.)\ ; \\ B \notin s &\Leftrightarrow \neg B \in s \text{ par maximalité}; \\ V(\neg B, s) = \mathbf{V} &\Leftrightarrow \neg B \in s \text{ par transitivité de } \Leftrightarrow. \end{aligned}$$

2. Clause pour la disjonction :

   (6.31) $V(B \vee C, s) = \mathbf{V} \Leftrightarrow B \vee C \in s$.

   (a) $B \vee C \in s \Rightarrow V(B \vee C, s) = \mathbf{V}$.

   Démonstration. Supposons que $B \vee C \in s$, alors en vertu des équivalences remarquables, on a $B \in s$ ou $C \in s$, par hypothèse inductive. Or, $V(B, s) = \mathbf{V}$ ou $V(C, s) = \mathbf{V}$. Dans les deux cas, $V(B \vee C, s) = \mathbf{V}$ en vertu des tables de vérité.

   (b) $V(B \vee C, s) = \mathbf{V} \Rightarrow B \vee C \in s$.

   On prouve la contraposée.

   Supposons que $B \vee C \notin s$,
   $\neg(B \vee C) \in s$ par les équivalences remarquables,
   $\neg B \wedge \neg C \in s$,
   $\neg B \in s$ et $\neg C \in s$,
   $V(\neg B, s) = \mathbf{V}$ et $V(\neg C, s) = \mathbf{V}$ par (6.30),

$V(\neg B \wedge \neg C) = \mathbf{V}$,
$V\neg(B \vee C) = \mathbf{V}$,
$V(B \vee C) \neq \mathbf{V}$,
$(B \vee C) \notin s \Rightarrow V(B \vee C) \neq \mathbf{V}$ en déchargeant l'hypothèse,
$V(B \vee C, s) = \mathbf{V} \Rightarrow B \vee C \in s$.

3. Clause pour l'opérateur de nécessité :

(6.32) $V(\Box A, s) = \mathbf{V} \Leftrightarrow \Box A \in s$.

Démonstration.

$$\begin{aligned} V(\Box A, s) = \mathbf{V} &\Leftrightarrow \forall t\ (sRt \Rightarrow V(A,t)) = \mathbf{V} \text{ par définition de } \Box; \\ &\Leftrightarrow V(A,t) = \mathbf{V} \Leftrightarrow A \in t\ (H.I.); \\ (6.33) \qquad &\Leftrightarrow \forall t\ (sRt \Rightarrow A \in t) \text{ par échange de conséquents} \\ &\Leftrightarrow \Box A \in s \text{ par le lemme préparatoire.} \end{aligned}$$

Regroupant (6.30), (6.31) et (6.32), on a :

(6.34) $\forall s \in \mathbf{S},\ V(A,s) = \mathbf{V} \Leftrightarrow A \in s$.

6.6.8.10. *Le théorème de complétude*

Partant du lemme d'extension des valuations :

$$\forall s \in \mathbf{S},\ V(A,s) = \mathbf{V} \Leftrightarrow A \in s,$$

on applique à ce lemme le théorème de la logique du premier ordre $\forall x\,(Fx \supset Gx) \supset (\forall x Fx \supset \forall x Gx)$ qui autorise de distribuer le quantificateur universel sur l'implication; on obtient :

(6.35) $\forall s \in \mathbf{S},\ V(A,s) = \mathbf{V} \Leftrightarrow \forall s \in \mathbf{S},\ A \in s$.

Lorsque l'ensemble Λ est vide, le lemme de dérivabilité se réduit à :

(6.36) $\forall s \in \mathbf{S},\ A \in s \Leftrightarrow \vdash A$.

De (6.35) et de (6.36) par syllogisme, on tire :

(6.37) $\forall s \in \mathbf{S},\ V(A,s) = \mathbf{V} \Leftrightarrow \vdash A$.

Le membre de gauche de l'équivalence dit que A est valide (vrai dans tous les mondes) d'où, on peut réécrire (6.37) ainsi :

(6.38) $\models A \Leftrightarrow \vdash A$.

6.6.8.11. *Adéquation faible et adéquation forte*

Prouver qu'à toute *formule valide* dans une classe de structures correspond un *théorème* d'une logique L, c'est établir l'adéquation faible. Prouver qu'à toute *relation logique de conséquence* entre ensemble de formules et une formule

– que celle-ci soit la relation sémantique de conséquence globale ou locale – correspond une *relation syntaxique de dérivabilité* dans L, c'est établir l'adéquation forte. La preuve d'adéquation forte est le point culminant de l'entreprise logique.

Dans le formalisme de Fitting, l'adéquation forte se note :

$$\Lambda \models_{g(\mathcal{F})} \Gamma \to B \text{ si et seulement si } \Lambda \vdash \Gamma \to B.$$

6.6.8.12. *Application de la méthode de preuve par les modèles canoniques à d'autres systèmes*

On a énuméré précédemment les propriétés de la relation d'accessibilité qui correspondent à dix axiomes (§ 2.2.5). Outre son intérêt intrinsèque, la correspondance trouve des applications dans les preuves de complétude. Elle indique les modifications à apporter à la preuve de complétude de K par modèle canonique pour obtenir une preuve de complétude par modèle canonique pour une autre logique (parmi les quinze mentionnées au chapitre 2).

Comme on l'a vu, tout système canonique est complet. On peut donc prouver la complétude en prouvant la canonicité. Pour prouver qu'un système d'axiomes est canonique, on cherche une propriété exprimable en logique du premier ordre qui correspond à l'axiome et on montre que la structure canonique du système S possède cette propriété. C'est là une application importante de la *théorie de la correspondance.*

Chapitre 7
Théorie des modèles. Deuxième partie

7.1. Filtration et modèles finis

7.1.1. *La filtration*

Comme on l'a vu dans le chapitre précédent, pour prouver que la logique L est complète relativement à la classe des structures $C(\mathcal{F})$ – c'est-à-dire que toutes les formules valides dans $C(\mathcal{F})$ sont des théorèmes de L – on prouve la contraposée, à savoir que si A n'est pas un théorème de L, alors il y a un point dans le modèle canonique de L qui rend fausse la formule A.

Dans son rôle de modèle falsificateur pour un non-théorème particulier A, Goldblatt observe ([Goldblatt 92], p. 31) que le modèle canonique $\mathcal{M}^A$ nous fournit une masse d'information superflue. La seule chose dont nous ayons besoin pour calculer la valeur de vérité de A en un point de $\mathcal{M}^A$, c'est de connaître la valeur de vérité dans $\mathcal{M}^A$ de tous les membres de l'ensemble des sous-formules de A, lesquelles sont en nombre fini. Or, le modèle canonique assigne une valeur de vérité à toutes les formules de L.

Les points qui diffèrent entre eux uniquement par la valeur de vérité qu'ils assignent à d'autres formules que les membres de l'ensemble des sous-formules de A diffèrent d'une manière négligeable compte tenu du but poursuivi (falsifier A). Nous pouvons donc les identifier et réduire le modèle falsificateur à un sous-modèle de celui-ci. Cette opération de réduction s'appelle filtration. La filtration permet de représenter une information infinie de manière finie.

7.1.2. *Notions préliminaires*

Définition 7.1. *L'ensemble $Sf(A)$ des sous-formules d'une formule A*, écrit

en notation primitive (c'est-à-dire sans $\vee, \supset$ et $\Diamond$), est défini ainsi :

$Sf(p) = \{p\}$;

$Sf(\neg A) = \{\neg A\} \cup Sf(A)$;

$Sf(A \wedge B) = \{A \wedge B\} \cup Sf(A) \cup Sf(B)$;

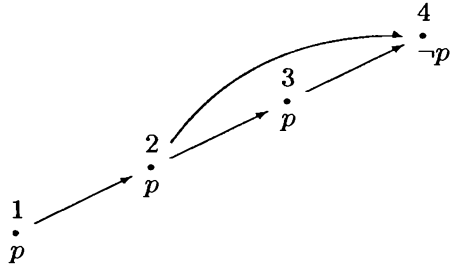
$Sf(\Box A) = \{\Box A\} \cup Sf(A)$.

Définition 7.2 : Equivalence des mondes relativement à un ensemble de formules. Deux mondes w et w' sont *équivalents relativement à l'ensemble* Φ *de formules* pour un modèle $\mathcal{M} = <W, R, V>$ si et seulement si pour toute formule B membre de Φ, B est vraie (respectivement fausse) dans w si et seulement si B est vraie (respectivement fausse) dans w'. Formellement :

$$w \approx_\Phi w' \text{ si et seulement si } \forall B \in \Phi,\ \models_w B \Leftrightarrow \models_{w'} B.$$

Cette relation d'équivalence s'appelle relation de filtration.

Exemple. Soit $\Phi = \{\Box p, p\}$. Dans le modèle suivant de K, p est mis pour $V(p) = \mathbf{V}$ et $\neg p$ pour $V(p) = \mathbf{F}$.



$$\models_{w_1} p,\ \models_{w_2} p,\ \models_{w_3} p,\ \not\models_{w4} p$$

$$\models_{w_1} \Box p,\ \not\models_{w_2} \Box p,\ \not\models_{w_3} \Box p,\ \models_{w_4} \Box p$$

$$w_1 \approx_p w_2 \approx_p w_3;\ w_1 \approx_{\Box p} w_4;\ w_2 \approx_{\{p, \Box p\}} w_3$$

Remarque. Le monde w_4 rend vrai $\Box p$ parce que $\Box p$ signifie "si w' est accessible à partir de w, $\models_{w'} p$" et que l'antécédent est trivialement vrai puisqu'il n'y a pas de monde accessible à partir de w_4. Le monde w_4 est un "dernier monde". Dans un modèle de D, modèle où la relation R est reproductive, w_4 ne serait pas un dernier monde (D n'en comporte pas).

Définition 7.3 : La classe d'équivalence modulo Φ. En divisant l'ensemble W par la relation d'équivalence $\approx_\Phi$, on obtient l'ensemble quotient $W_{\approx_\Phi}$ formé d'ensembles disjoints non vides que l'on appelle *classes d'équivalence.* On les note par $[w]_\Phi$. Dans l'exemple, l'ensemble des classes d'équivalence, c'est-à-dire $\{[w]_\Phi | w \in W\}$ est {{1,3} , {2} ,{4}}.

Définition 7.4 : La filtration à travers un ensemble. Soit un modèle $\mathcal{M} = <W, R, V>$ et Φ une classe de formules, $\mathcal{M}^* = <W^*, R^*, V^*>$ est une filtration à travers Φ si $\mathcal{M}^*$ remplit les conditions suivantes :

(1) Les mondes de W^* sont les classes d'équivalence obtenues en divisant W par la relation de filtration. Formellement :

$$W^* = \{[w]_\Phi | w \in W\}.$$

(2) La valuation V^* est la restriction de la valuation V aux membres de W^*. Plus précisément, la valeur d'un atome propositionnel p est l'ensemble des classes d'équivalence telles que :

$$V^*(p) = \{[w]_\Phi | \mathcal{M} \models_v p\}.$$

(3) La relation R^* remplit les deux conditions suivantes :

(i) Si sRt, alors $[s]R^*[t]$.

(ii) Si $[s]R^*[t]$, alors pour tout $\Box A \in \Phi, \mathcal{M} \models_s \Box A \Rightarrow \mathcal{M} \models_t A$.

7.1.3. *Théorème de préservation de la valuation*

Théorème 7.1. Soit $\mathcal{M}$ un modèle de K et $\mathcal{M}^*$ une filtration de $\mathcal{M}$ modulo Φ, alors pour toute formule A :

$$\mathcal{M}^* \models_{[s]} A \Leftrightarrow \mathcal{M} \models_s A.$$

Le théorème énonce que la correspondance entre V^* et V n'existe pas seulement au niveau des atomes propositionnels mais qu'elle s'étend à toutes les formules A de la logique L [Rautenberg 79], p. 204.

Démonstration. La preuve se fait par induction. Le cas de base où $A = p$ est vrai par définition. Considérons seulement le cas où $A = \Box B$. Traitons d'abord la direction $\Rightarrow$:

$$7.1.1 \ : \ \mathcal{M}^* \models_{[s]} \Box B \Rightarrow \mathcal{M} \models_s \Box B.$$

(1) $\mathcal{M}^* \models_{[s]} \Box B$ par hypothèse (antécédent de 7.1 pour $A = \Box B$).

(2) $[s]R^*[t] \Rightarrow \mathcal{M}^* \models_{[t]} B$ par définition de $\Box$.

(3) Supposons que sRt.

(4) Alors $[s]R^*[t]$ en vertu de la définition 7.4(3)(i).

(5) Donc $\mathcal{M}^* \models_{[t]} B$ par *modus ponens* sur (2) et (4).

(6) $\mathcal{M}^* \models_{[t]} B \Leftrightarrow \mathcal{M} \models_t B$ (par *H.I*).

(7) $\mathcal{M} \models_t B$ de (5) et (6) par échange d'équivalents.

(8) $sRt \Rightarrow \mathcal{M} \models_t B$ en déchargeant l'hypothèse (3).

(9) $\mathcal{M} \models_s \Box B$ de (8) par définition de $\Box$.

(10) 7.1.1 en déchargeant (1).

Traitons ensuite la direction $\Leftarrow$:

$$7.1.2 : \mathcal{M} \models_s \Box B \Rightarrow \mathcal{M}^* \models_{[s]} \Box B.$$

(1) $\mathcal{M} \models_s \Box B$ par hypothèse (conséquent de de 7.1 pour $A = \Box B$).

(2) Supposons que $[s]R^*[t]$.

(3) Pour tout $\Box B \in \Phi, \mathcal{M} \models_s \Box B \Rightarrow \mathcal{M} \models_t B$ par la définition 7.4(3)(ii).

(4) $\mathcal{M} \models_t B \Leftrightarrow \mathcal{M}^* \models_{[t]} B$ (par *H.I*).

(5) Pour tout $\Box B \in \Phi, \mathcal{M} \models_s \Box B \Rightarrow \mathcal{M}^* \models_{[t]} B$ de (3) et (4) par échange d'équivalents.

(6) $\mathcal{M}^* \models_{[t]} B$ de (1) et de (5) par *modus ponens.*

(7) $[s]R^*[t] \Rightarrow \mathcal{M}^* \models_{[t]} B$ en déchargeant (2).

(8) $\mathcal{M}^* \models_{[s]} \Box B$ par définition de $\Box$.

(9) 7.1.2 en déchargeant (1).

Une objection. Un modèle filtré est un modèle réduit, donc un modèle beaucoup plus aisé à explorer lorsque nous cherchons à "falsifier" A. Mais ce modèle réduit ne risque-t-il pas de nous priver de la possibilité de falsifier une formule qui était falsifiable dans le modèle initial ? L'objection est écartée dans le théorème suivant.

7.1.4. *Préservation de la non-validité*

Théorème 7.2. Si la formule A est non valide dans un modèle $\mathcal{M} = <W, R, V>$, elle le reste dans toutes les filtrations $\mathcal{M}^* = <W^*, R^*, V^*>$ modulo $Sbf(A)$.

Démonstration. ([Hughes and Cresswell 84], p. 140) Supposons que s falsifie la formule A dans le modèle $\mathcal{M}$, ce qui se note $\mathcal{M} \not\models_s A$. Par la définition de W^*, il existe un $[s]$ tel que $s \approx_{Sbf(A)} [s]$. Or, en vertu du théorème précédent, $\mathcal{M}^* \models_{[s]} A \Leftrightarrow \mathcal{M} \models_s A$, donc $\mathcal{M}^* \not\models_{[s]} A$.

7.1.5. *La propriété du modèle fini (p.m.f.)*

Si Φ contient un nombre fini de formules, toute filtration à travers Φ nous donnera un modèle fini. En effet, la relation d'équivalence donne lieu à 2^n classes d'équivalences distinctes [Jansana 90], p. 84.

Pour qu'une logique L ait la propriété du modèle fini, il ne suffit pas que tout non-théorème de L soit faux dans un modèle fini. Il faut, en outre, que le modèle fini en question soit un modèle de L.

Toutes les filtrations ne satisfont pas cette condition : la filtration d'un modèle peut faire perdre à la structure sur laquelle il est basé certaines propriétés telle que, par exemple, la transitivité.

Pour prouver que $S4$ possède la propriété d'avoir un modèle fini, il n'est nullement nécessaire que toutes les filtrations du modèle canonique de $S4$ soient réflexives et transitives. Il suffit de prouver qu'il existe au moins une filtration $\mathcal{M}^*$ à travers Φ qui est réflexive et transitive [Hughes and Cresswell 84], p. 143.

Trouver la "bonne relation de filtration" n'est donc pas un travail de routine. Pour ce qui est de la transitivité, la relation de filtration (due à Lemmon) est :

$[s]R^{\tau}[t]$ ssi pour tout B, si $\Box B \in \Phi$ et $\mathcal{M} \models_s \Box B$ alors $\mathcal{M} \models_t \Box B \wedge B$.

On verra dans les règles de la preuve par tableaux et dans celles de la déduction naturelle une contrainte syntaxique qui correspond à cette condition sémantique.

7.1.6. *La propriété de la structure finie (p.s.f.)*

Définition 7.5. Si L est un système modal normal, L a la *propriété de la structure finie* si et seulement si pour chaque formule B qui n'est pas un théorème de L, il existe une structure $\mathcal{F} = <W, R>$ dans laquelle W est un ensemble fini et :

(i) il existe un modèle $\mathcal{M} = <W, R, V>$ basé sur $\mathcal{F}$ dans lequel pour au moins un monde w, $\mathcal{M} \not\models_w B$;

(ii) chaque théorème de L est valide dans $\mathcal{F}$ [Hughes and Cresswell 84], p. 150.

7.1.7. *Modèles finis, structures finies et décidabilité*

On démontre que tout système qui a la *propriété du modèle fini* a la *propriété de la structure finie* et réciproquement. La preuve est donnée dans [Hughes and Cresswell 84], p. 150-152.

Quant au lien entre la *propriété de la structure finie* et la *décidabilité*, on l'établit aisément par le raisonnement qui suit.

Si $\mathcal{F}$ est une structure finie, il est évident que nous disposons d'une procédure finie pour vérifier si toutes les formules membres de l'ensemble fini d'axiomes α sont valides dans $\mathcal{F}$.

En effet ([Hughes and Cresswell 84], p. 153), il existe une procédure effective pour engendrer toutes les structures finies dans un certain ordre et donc aussi pour engendrer de manière effective toutes les *structures pour* L, c'est-à-dire toutes les structures dans lesquelles les axiomes de L sont valides. On peut, en effet, vérifier en un temps fini si une structure $\mathcal{F}$, candidate au rôle de *structure pour* L, remplit bien ce rôle.

Si la formule A de L n'est pas un théorème de L, elle sera non valide dans l'une des *structures finies pour* L. On a donc une procédure effective pour

repérer les non-théorèmes. Ce qui ne veut pas dire encore que l'algorithme soit performant !

Pour que le système L soit décidable, il faut, par surcroît, que l'on puisse aussi, en un nombre fini de pas, repérer les théorèmes. Nous disposons d'un tel algorithme, mais il diffère du premier. La procédure, cette fois, consiste à engendrer toutes les preuves dans un certain ordre (I, p. 90, 91).

7.2. La théorie de la correspondance

7.2.1. *La traduction de la logique modale propositionnelle dans le calcul des prédicats*

On peut définir récursivement une fonction de traduction τ qui associe, à chaque formule propositionnelle évaluée dans un monde, une formule du calcul des prédicats.

Définition 7.6.

1. $\tau(\top, x) = \top$, $\tau(\bot, x) = \bot$.
2. $\tau(p, x) = Px$.

   [Ce qui se lit : "la traduction de l'atome propositionnel p évalué dans le monde x est la formule ouverte Px formée du prédicat unaire P combiné à la variable individuelle x par la relation syntaxique de prédication".]
3. $\tau(\neg A, x) = \neg\tau(A, x)$.
4. $\tau(A \supset B, x) = \tau(A, x) \supset \tau(B, x)$.
5. $\tau(\Box A, x) = \forall x' \, [xRx' \supset \tau(A, x')]$.
6. $\tau(\Diamond A, x) = \exists x' \, [xRx' \wedge \tau(A, x')]$.

Exemple. Appliquons l'algorithme de traduction à l'axiome 4 :

$$\tau(\Box p \supset \Box\Box p, x).$$

1. $\underbrace{\tau(\Box p, x)}_{a} \supset \underbrace{\tau(\Box\Box p, x)}_{b}$.

2a. $\forall y \, (xRy \supset \tau(p, y))$ de 1a par la clause 4.

3a. $\forall y \, (xRy \supset Py)$ de 2a par la clause 2.

4b. $\forall y \, (xRy \supset \tau(\Box p, y))$ de 1b par la clause 4.

5b. $\forall y \, (xRy \supset \forall z(yRz \supset \tau(p, z)))$ de 4b par la clause 4.

6b. $\forall y \, (xRy \supset \forall z(yRz \supset Pz))$ de 5b par la clause 2.

7b. $\forall y \, (xRy \supset \forall x(yRx \supset Px))$ de 6b par renommage de variables liées.

8b. $\forall y \, (xRy \supset Py) \supset \forall y \, (xRy \supset \forall x \, (yRx \supset Px))$ de 3a et 7b.

7.2.2. *Conséquences de la possibilité de traduire la logique modale propositionnelle en logique du premier ordre*

Théorème 7.3. A est valide dans K si et seulement si la formule A' de la logique du premier ordre (FO) qui en est la traduction est valide [Fitting 93b], p. 104.

Pour démontrer un théorème de K, il suffira donc de le traduire au moyen de τ en FO et d'appliquer les méthodes de preuve pour la logique du premier ordre et notamment les démonstrateurs automatiques de théorèmes.

Plus généralement, la traductibilité d'une logique modale L dans la logique du premier ordre autorise l'extension à L de résultats de la logique du premier ordre tel que le théorème de compacité (I, p. 411, 426), ou le théorème de Löwenheim-Skolem qui énonce que tout ensemble de formules de la logique du premier ordre qui est sémantiquement consistant, est sémantiquement consistant dans un *domaine dénombrable.*

7.2.3. *Limites de la traduction dans la logique du premier ordre*

Pour comprendre la portée exacte de cette traduction, il convient de prendre en compte ses limites.

En premier lieu, elle ne permet de faire usage des méthodes de preuve de la logique du premier ordre pour prouver une formule propositionnelle modale que pour certaines logiques modales (notamment les quinze logiques modales classiques).

Certaines logiques ne se prêtent pas à ce traitement. Prenons par exemple l'axiome de Mc Kinsey ($\Box\Diamond A \supset \Diamond\Box A$). On peut le traduire en FO grâce à l'algorithme τ. Si ensuite on tente de le démontrer par la méthode de résolution (I, p. 279-286), on doit nier la formule qui résulte de la traduction en logique du premier ordre, mettre celle-ci en forme clausale de Skolem et essayer d'arriver à une contradiction. Or, on aboutit à un blocage : la mise en forme de Skolem exige le recours à des quantificateurs branchés dont l'emploi nous fait sortir du champ de la logique du premier ordre [Bläsius und Bürckert 92], p. 256-257, pour entrer dans celui d'une logique d'ordre supérieur.

En second lieu, il importe d'observer que la logique modale traduite dans la logique du premier ordre n'est pas une simple *variante notationnelle* de la logique du premier ordre. Elle diffère de la logique du premier ordre standard par la nature des *objets* sur lesquels elle quantifie. Ceux-ci sont des mondes, états, situations plutôt que des individus quelconques. Mais plus fondamentalement, elle en diffère en cela que les objets sur lesquels elle quantifie ne peuvent être *choisis au hasard* dans le domaine de variation de la variable. En logique modale, les mondes, situations, etc ... sur lesquels on quantifie doivent être *reliés* par la relation d'accessibilité.

C'est ce caractère de *localité* qui fait la spécificité de la logique modale et qui est à l'origine des bonnes propriétés, que possèdent notamment les quinze

logiques modales étudiées au deuxième chapitre, telle que la décidabilité qui manque à la logique du premier ordre, laquelle n'est que semi-décidable.

En troisième lieu, pour démontrer la validité d'une formule A dans une logique autre que K – mettons T – il ne suffit pas de traduire A dans la logique du premier ordre. Il faut, par surcroît, découvrir la formule du premier ordre qui *correspond* à l'axiome T : " ... pour prouver que X est valide dans T, il faut montrer que $(\forall x_1)R(x_1, x_1) \supset (\forall x_1)\tau(X, x_1)$ est valide classiquement [selon FO]" [Fitting 93], p. 404.

7.2.4. *Comment établir un résultat de correspondance ?*

La simple inspection de la formule $\Box p \supset \Box\Box p$ – nous nous limiterons à cet exemple – ne montre pas qu'elle est vérifiée dans un modèle de Kripke *si et seulement si* la relation d'accessibilité R est transitive. Trouver cette correspondance exige une démarche inventive.

Une fois formulée la thèse de la correspondance entre $\Box p \supset \Box\Box p$ et la transitivité de la relation d'accessibilité, il reste à démontrer cette thèse.

Démontrons d'abord que si $\Box p \supset \Box\Box p$ est valide dans une structure $\mathcal{F} = <W, R>$, la relation est transitive. Supposons qu'on ait une structure dans laquelle de w_0 on accède à w_1 et de w_1 à w_2 (voir la figure 7.1). Construisons le modèle $\mathcal{M} = <W, R, V>$ en prenant n'importe quelle valuation de p telle que $V(p) = \{x \in X \mid w_0 R x\}$.

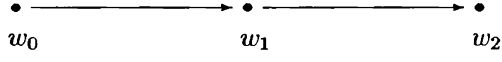


Figure 7.1 *Exemple de structure*

Dans $\mathcal{M}$, il est clair que l'on a $\models_{w_0} \Box p$. En vertu de l'axiome $\Box p \supset \Box\Box p$, on a aussi $\models_{w_0} \Box\Box p$. D'où on a $\models_{w_1} \Box p$ et $\models_{w_2} p$. On peut donc prendre w_2 comme valeur pour x dans la valuation $V(p)$, ce qui donne $w_0 R w_2$. Donc, la relation est transitive.

Démontrons à présent que si la relation R est transitive, $\Box p \supset \Box\Box p$ doit être valide. Supposons R transitive et $\Box p \supset \Box\Box p$ non valide. On a dans ce cas la structure représentée par la figure 7.2.

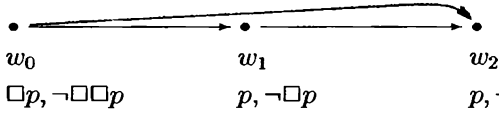


Figure 7.2 *Structure transitive*

On a $\models_{w_0} \neg\Box\Box p$ qui équivaut à $\models_{w_0} \Diamond\neg\Box p$. Il doit exister un monde w_i accessible à partir de w_0 tel que $\models_{w_i} \neg\Box p$, c'est-à-dire tel que $\models_{w_i} \Diamond\neg p$. Ce

monde w_i ne peut être w_2 car w_2 est un "dernier monde" d'où on ne peut aller vers un monde w_j qui vérifie $\neg p$. Donc ce monde w_i doit être w_1. Mais alors w_2 contient à la fois p (à cause de $\Box p$ en w_0 et de la transitivité) et $\neg p$ (à cause de $\Diamond \neg p$ en w_1), ce qui est impossible.

7.2.5. *L'algorithme de Sahlqvist-van Benthem*

Pour une certaine classe de formules modales propositionnelles, il existe un algorithme qui permet de convertir les formules appartenant à cette classe en formules de la logique du premier ordre qui énoncent les conditions nécessaires et suffisantes que doit remplir la relation d'accessibilité pour vérifier les formules de logique modale considérées.

Le fragment du langage modal couvert par l'algorithme est la classe obtenue à partir d'une classe (1) que l'on étend progressivement en (2) et (3).

(1) La classe de départ contient les *formules très simples de Sahlqvist*, à savoir des implications de la forme $A \supset B$ où l'antécédent A est construit à partir de $\top$, $\bot$ et d'atomes propositionnels en utilisant uniquement $\wedge$ et $\Diamond$. Une formule très simple de Sahlqvist est une implication $A \supset B$ dans laquelle B est une formule positive (toutes les occurrences de p sont dans la portée d'un nombre pair de négations) et l'antécédent est construit comme on a dit.

(2) Une *formule simple de Sahlqvist* est une implication $A \supset B$ dans laquelle B est positive et où l'antécédent est construit à partir de $\top$, $\bot$, d'atomes préfixés par l'opérateur $\Box$ en utilisant uniquement $\wedge$ et l'opérateur $\Diamond$.

(3) Une *formule de Sahlqvist* est une implication $A \supset B$ où le conséquent B est positif et dont l'antécédent est construit à partir de $\top$, $\bot$, d'atomes précédés d'opérateurs $\Box$ et de formules négatives en utilisant $\wedge$, $\vee$ et l'opérateur $\Diamond$.

7.2.6. *Notions préliminaires*

Avant de présenter une application de l'algorithme de Sahlqvist-van Benthem [Sahlqvist 75, van Benthem 76], il convient de donner les définitions et le lemme suivants [Blackburn et al.] (à paraître).

Définition 7.7.

a. La formule A est positive pour l'atome propositionnel p si la traduction de A en logique du premier ordre est positive pour ce qui est du prédicat unaire $P(\ )$ correspondant.

b. Si la formule A est positive (respectivement négative) en p, alors $\neg A$ est négative (respectivement positive) en p.

Définition 7.8.

a. La formule A est monotone vers le haut si sa vérité est préservée quand on étend la valuation (c'est-à-dire quand on étend la classe de mondes possibles qui vérifient p).

b. La formule A est monotone en p vers le bas si sa vérité est préservée quand on rétrécit la valuation de p.

On relie ensuite la notion *syntaxique* de *positivité* (respectivement de négativité) à la notion *sémantique* de *monotonie* vers le haut (respectivement vers le bas) par un lemme.

Lemme 7.4. Si la formule A est positive en p, elle est monotone vers le haut. Si elle est négative en p, elle est monotone vers le bas.

7.2.7. *La traduction en logique du deuxième ordre*

Les paragraphes 2.2.4 et 2.2.5 présentent des schémas de formules qui sont associées à certaines propriétés de la relation d'accessibilité. L'algorithme de Sahlqvist-van Benthem permet de découvrir mécaniquement les formules du premier ordre qui correspondent à ces schémas.

La première phase de l'exécution de l'algorithme précité consiste cependant dans la traduction en *logique du deuxième ordre* de la formule propositionnelle dont on cherche le correspondant en logique du premier ordre.

La raison de ce détour par le deuxième ordre s'explique aisément. On cherche à caractériser la relation d'accessibilité et à définir des propriétés de structures. Or la notion de structure est une notion du deuxième ordre. Rappelons-nous qu'une propriété appartient à une structure $\mathcal{F} = < W, R >$ si elle est possédée par tous les modèles $\mathcal{M} = <W, R, V>$ qui sont basés sur la structure $\mathcal{F}$. La caractérisation d'une propriété d'une structure fait donc usage d'une quantification de second ordre : on quantifie sur les valuations V, c'est-à-dire sur des sous-ensembles de l'ensemble W.

La traduction d'une formule propositionnelle en formule de la logique du deuxième ordre fait appel à l'algorithme de traduction τ' qui diffère de l'algorithme τ uniquement en ceci : des constantes prédicatives unaires sont désormais remplacées par des variables prédicatives unaires. Par exemple, la traduction τ' de $\Box\Box p \supset \Box p$ en logique du deuxième ordre sera :

$$\forall P\,(\forall y\,(xRy \supset Py) \supset \forall y\,(xRy \supset \forall x\,(yRx \supset Px))).$$

7.2.8. *L'instantiation minimale*

La traduction de la logique propositionnelle en logique du deuxième ordre n'est qu'une étape dans l'exécution de l'algorithme. Comme la formule qui correspond – au sens technique – à la formule propositionnelle est une

formule du premier ordre, il va falloir *instancier* la formule du deuxième ordre obtenue par τ'. Cette instantiation n'est pas quelconque. On doit prendre l'instantiation *minimale* qui rende vrai *l'antécédent* de l'implication $A \supset B$ qui est le connecteur principal des formules figurant dans le fragment de langage de Sahlqvist-van Benthem. Voyons en quoi consiste l'instantiation minimale.

Il y a trois cas à considérer :

(1) La variable prédicative P n'apparaît que dans l'antécédent de l'implication $A \supset B$. Dans ce cas, on l'instancie au moyen d'une constante prédicative qui désigne l'ensemble vide, à savoir la constante $\lambda u.u \neq u$.

Rappelons qu'il s'agit d'un prédicat en notation lambda que l'on transforme ensuite en prédicat en notation ordinaire au moyen d'une opération de lambda conversion illustrée ci-dessous. On biffe λu et on substitue a aux occurrences de u :

$$\lambda u.(u \neq u)a;$$

$$a \neq a.$$

(2) La variable prédicative P apparaît à la fois dans l'antécédent et dans le conséquent mais elle n'apparaît qu'une seule fois dans l'antécédent de $A \supset B$. Dans ce cas, on instancie P au moyen de la constante $\lambda u.(u = x)$.

(3) La variable prédicative P apparaît à la fois dans l'antécédent et dans le conséquent de $A \supset B$. Dans l'antécédent, elle apparaît un nombre k de fois. Dans ce cas, on instancie P au moyen de la constante $\lambda u.(u = x_1 \vee \ldots \vee u = x_k)$. On verra plus loin la raison d'être des choix (2) et (3).

7.2.9. *Exemple d'application de l'algorithme*

Considérons la formule $\Diamond p \supset \Diamond\Diamond p$. Elle appartient à la classe des "formules très simples de Sahlqvist". On peut donc lui appliquer l'algorithme pour la convertir en une formule du premier ordre qui lui corresponde.

La première étape est la traduction par τ' de cette formule de la logique propositionnelle en logique du deuxième ordre au moyen des transformations successives énumérées ci-dessous :

$$\forall V\, \forall w\, \mathcal{F}, V \models_w \Diamond p \supset \Diamond\Diamond p;$$

$$\forall V\, \forall w\, (V \models_w \Diamond p \Rightarrow V \models_w \Diamond\Diamond p);$$

$$\forall V\, [\exists w'\, (wRw' \wedge \models_{w'} p) \Rightarrow \exists x\, \exists y\, (wRx \wedge xRy \wedge \models_y p)].$$

Sachant que $\models_{w'} p$, $w' \in V(p)$ et $P(w')$ sont trois variantes notationnelles qui expriment le même choix, on arrive à :

$$\forall P\, [\exists w'\, (wRw' \wedge Pw') \Rightarrow \exists x\, \exists y\, (wRx \wedge xRy \wedge Py)].$$

La deuxième étape est l'application des lois de passage pour faire sortir de l'antécédent les quantificateurs existentiels introduits lors de la traduction d'opérateurs $\Diamond$:

$$\forall P\, \forall w' \, [(wRw' \wedge Pw') \Rightarrow \exists x\, \exists y\, (wRx \wedge xRy \wedge Py)].$$

L'antécédent contient une partie relationnelle – ici wRw' – et une partie atomique – ici Pw' – c'est-à-dire un prédicat unaire qui est le témoin d'un atome propositionnel figurant dans la formule de départ.

Enfin, par définition, une formule de Sahlqvist-van Benthem a un conséquent positif. Pour exprimer ces trois points dans le formalisme, on dit que la formule précédente a la forme :

$$\forall P_1 \ldots \forall P_n\, \forall w_1 \ldots \forall w_m\, (\mathrm{REL} \wedge \mathrm{AT} \Rightarrow \mathrm{POS}),$$

avec dans le cas présent $n = 1$ et $m = 1$. (REL, AT et POS sont les abréviations de "relationnel", "atomique" et "positif".)

La troisième étape consiste dans le choix d'une *instantiation minimale* pour la variable P. Sachant que la formule traitée ici tombe alors dans la catégorie (2) de la définition d'"instance minimale", on instanciera P par $(\lambda u.u = b)$. Appliquant la lambda conversion, on obtient la formule :

$$(aRb \wedge b = b) \supset \exists x\, \exists y\, (aRx \wedge xRy \wedge y = b).$$

La quatrième et dernière étape consiste à effacer les tautologies qui résultent de l'instantiation minimale et à simplifier le conséquent sur base de la logique de l'identité, ce qui donne :

$$aRb \supset \exists x\, (aRx \wedge xRb).$$

Remarque. Le choix des instantiations minimales (2) et (3) a été dicté par la préoccupation d'obtenir des tautologies éliminables à la place de AT.

On est arrivé *par un calcul* à la formule de la logique du premier ordre qui correspond, au sens technique de ce terme, à la formule modale propositionnelle :

$$\Diamond p \supset \Diamond\Diamond q,$$

qui est la duale de :

$$\Box\Box p \supset \Box p.$$

On serait arrivé au même résultat en travaillant sur cette dernière formule.

7.2.10. *Justification de l'algorithme*

Pour qu'une formule A' – ici $aRb \supset \exists x\,(aRx \wedge xRb)$ – soit la correspondante en logique du premier ordre d'une formule A de la logique propositionnelle – ici $\Diamond p \supset \Diamond\Diamond p$ –, une condition nécessaire, mais non suffisante, doit être remplie. Il faut que les deux formules soient équivalentes. Elles sont en effet

deux formulations, dans des formalismes différents, de la même propriété de la relation d'accessibilité R dans la structure $\mathcal{F} = <W, R>$, à savoir la densité.

Or on a fait un usage essentiel de l'instantiation dans le travail de conversion de $\Diamond p \supset \Diamond\Diamond p$ en $aRb \supset \exists x\ (aRx \wedge xRb)$. La règle d'instantiation universelle justifie l'implication de :

$$\forall P\ [(aRb \wedge Pb) \supset \exists x\ \exists y\ (aRx \wedge xRy \wedge Py)],$$

à :

$$((aRb \wedge \lambda y.(y = b)b) \supset \exists x\ \exists y\ (aRx \wedge xRy \wedge \lambda y.(y = b)y),$$

d'où l'on tire ensuite, par lambda conversion :

$$(aRb \wedge b = b) \supset \exists x\ \exists y\ (aRx \wedge xRy \wedge y = b).$$

Il reste à justifier la réciproque. Celle-ci n'est pas triviale. Il ne va pas de soi que l'on puisse déduire une proposition universelle d'une proposition singulière.

C'est à ce point précis qu'interviennent les notions d'*instantiation minimale*, d'*occurrence positive* et de *monotonie vers le haut*. En choisissant l'*instantiation minimale* qui rend vrai l'antécédent et en imposant aux implications de Sahlqvist-van Benthem d'avoir un conséquent qui soit *positif* et donc *monotone vers le haut*, on a créé les conditions qui assurent la légitimité de l'implication réciproque. En effet, si l'évaluation obtenue par l'"instantiation minimale" est la plus petite qui rende vrai l'antécédent et si, par ailleurs, le conséquent est monotone vers le haut, c'est-à-dire s'il reste vrai pour toute évaluation plus grande, alors aucune autre évaluation (aucune instantiation de $P_1 \ldots P_k$) ne peut rendre vrai l'antécédent et falsifier le conséquent. En d'autres termes, dans ces conditions, toutes les évaluations, c'est-à-dire tous les choix de P, rendent vraie l'implication $A \supset B$.

Rappelons qu'évaluer la variable P, c'est lui donner une valeur, c'est-à-dire en l'occurrence une classe de mondes possibles (un sous-ensemble de W) dans lesquels p est vrai. Rendre "plus grande" l'évaluation, c'est étendre la classe des mondes possibles donnée comme valeur à P en ajoutant des mondes.

7.2.11. *Portée et limites de l'algorithme*

Le choix de l'évaluation qui permet de trouver la formule du premier ordre qui correspond à une formule modale propositionnelle n'est pas un travail de routine, observe S. Popkorn [Popkorn 94], p. 62. L'invention d'un algorithme qui permet de trouver cette évaluation par un calcul est donc un progrès majeur.

Trouver pour un axiome d'un système donné de la logique modale propositionnelle la formule du premier ordre qui lui correspond est un préalable à la construction d'une preuve de complétude par les modèles canoniques. Le théorème de Chagrova, cependant, en indique les limites : la question de savoir si une formule modale de base arbitrairement choisie a un correspondant dans le premier ordre est indécidable.

Exercice. Le lecteur est invité à appliquer l'algorithme à $p \supset \Diamond p$ (duale de $\Box p \supset p$) et à $\Diamond\Diamond p \supset \Diamond p$ (duale de $\Box p \supset \Box\Box p$) pour retrouver mécaniquement les formules du premier ordre $\forall x\,(xRx)$ et $\forall x\,\forall y\,\forall z\,[(xRy \wedge yRz) \supset xRz]$.

7.3. Résultats de préservation

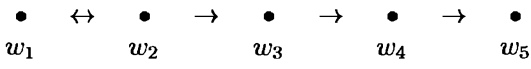
7.3.1. *Les sous-modèles engendrés*

Certaines transformations opérées sur la structure d'un modèle laissent intacte la classe des formules vraies dans ce modèle. L'une des fonctions de la théorie des modèles pour la logique modale est de décrire de telles transformations. Nous nous bornerons ici à étudier les transformations qui ont un intérêt particulier pour la théorie de la preuve, en particulier l'opération de formation de *sous-modèles engendrés* et la formation d'*ultraproduits*.

7.3.2. *Définitions préliminaires*

Définition 7.9. On appelle *structure engendrée* une structure dans laquelle il existe au moins un monde (le "générateur") qui est relié à chaque autre monde en un nombre n de pas [Hughes and Cresswell 84], p. 392. Le domaine des mondes W_w engendré par w se note : $\{v \in W | w\,R^n\,v\}$.

Exemple. La structure ci-dessous est une structure engendrée par 2 générateurs w_1 et w_2 :

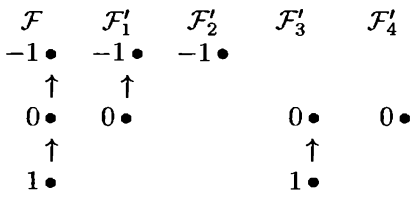


Définition 7.10. On appelle *sous-structure* de $\mathcal{F}$ engendrée par le générateur w^* une structure $\mathcal{F}'$ qui remplit les conditions :

1. $\mathcal{F}'$ est une structure telle que pour $\mathcal{F} = <W, R>$, $\mathcal{F}' = <W', R'>$ où $W' \subseteq W$ et R' est définie sur W' (c'est-à-dire restreinte à W', ce qui se note $R' = R \restriction W'$).
2. $\mathcal{F}'$ satisfait la condition d'hérédité. Intuitivement, cela revient à demander que W' soit un sous-ensemble de W *fermé vers le dessus*, ce qui se note :

$$\forall w \in W', \forall v \in W\,(\mathcal{M} \models wRv \Rightarrow w \in W').$$

Exemple. L'exemple est adapté de [Chagrov and Zakharyaschev 97], p. 69. Si $\mathcal{F} = <W, R>$ où $W = \{1; 0; -1\}$ et R la relation $<$, les sous-structures $\mathcal{F}'_1$ et $\mathcal{F}'_2$ remplissent la seconde condition, mais $\mathcal{F}'_3$ et $\mathcal{F}'_4$ ne le font pas.



Définition 7.11. On appelle *modèle engendré* une structure engendrée munie d'une valuation.

Définition 7.12. On appelle *sous-modèle engendré* une sous-structure engendrée munie d'une valuation V' qui remplit la condition suivante : la valuation V' de $\mathcal{M}' = <W', R', V'>$ est identique à la valuation V de $\mathcal{M}$, mais est restreinte au monde W' ce qui se note : $V' = V \restriction W'$.

7.3.3. *Intérêt de la notion de modèle et de sous-modèle engendrés*

Comme l'observe Chellas, "la valeur de vérité d'une formule A évaluée relativement à un monde possible w dans un modèle standard dépend seulement des sous-formules de A (incluant A elle-même) et des descendants des w par la relation R (incluant w lui-même). Un monde w' est descendant de w s'il peut être atteint à partir de w par un nombre fini de pas (fût-ce 0 pas) par la relation R. Les formules qui ne figurent pas dans la morphologie de A et les mondes w' qui ne descendent pas de w n'ont pas d'effet sur la valeur de vérité de A dans w" [Chellas 80], p. 95.

7.3.4. *Théorème de préservation pour les sous-modèles engendrés*

Théorème 7.5. Les sous-modèles engendrés préservent la valeur de vérité des formules modales des systèmes dont ils sont issus. Ce qui formellement se dit : Pour tout monde w' membre de W', et pour toute formule A :

$$V'(A, w') = V(A, w');$$

ou encore :

$$\mathcal{M}' \models_{w'} A \Leftrightarrow \mathcal{M} \models_{w'} A.$$

Dans la direction $\Rightarrow$, le théorème dit que les formules vraies dans le sous-modèle engendré $\mathcal{M}'$ restent vraies dans le modèle $\mathcal{M}$. Dans la direction $\Leftarrow$, il énonce que les formules modales vraies dans $\mathcal{M}$ restent vraies quand on restreint $\mathcal{M}$ à $\mathcal{M}'$. Dans cette dernière direction, le théorème est trivial et repose uniquement sur la notion de sous-modèle.

Le théorème se démontre par induction.

Base : en vertu des définitions préliminaires, pour tout monde w' membre de W' et pour tout atome propositionnel p, on a :

(7.1) $\quad V'(p, w') = V(p, w')$;

ou encore :

$$\mathcal{M}' \models_{w'} p \Leftrightarrow \mathcal{M} \models_{w'} p.$$

On étend cette propriété par induction à une formule A de complexité quelconque. Examinons seulement la clause inductive concernant l'opérateur modal $\Box$. Considérons $A = \Box B$ et prouvons la direction non triviale, à savoir que :

$$\mathcal{M}' \models_v \Box B \Rightarrow \mathcal{M} \models_v \Box B.$$

1. Supposons que $\mathcal{M}' \models_v \Box B$.
2. Supposons que $vR'v'$ où v' est membre de W' parce que $\mathcal{M}$ est un modèle engendré (propriété d'hérédité).
3. $vR'v' \Rightarrow \mathcal{M}' \models_{v'} B$ par définition de $\Box$.
4. $\mathcal{M}' \models_{v'} B$ par *modus ponens* (sur 2 et 3).
5. $\mathcal{M} \models_{v'} B$ de 4 par hypothèse inductive.
6. $vR'v' \Rightarrow \mathcal{M} \models_{v'} B$ en déchargeant l'hypothèse 2.
7. $vR'v' \Leftrightarrow vRv'$ en vertu de la définition de sous-modèle.
8. $vRv' \Rightarrow \mathcal{M} \models_{v'} B$ en échangeant l'antécédent de 6 contre son équivalent.
9. $\mathcal{M} \models_v \Box B$ par définition de $\Box$.
10. $1 \Rightarrow 9$ en déchargeant 1.

7.3.5. *Ultrafiltres et ultraproduits*

Dans la formulation du théorème de la déduction pour la logique modale, on utilise certains concepts d'algèbre qui sont introduits ci-dessous. Ils interviennent également dans la formulation d'un théorème du premier ordre qui peut être étendu à la logique modale, le théorème de Loš.

Définition 7.13 : Le produit cartésien généralisé. La définition du produit cartésien :

$$x(a, b) =_{def} \{x : \exists u \exists v [x = <u, v> \wedge u \in a \wedge v \in b]\},$$

ne s'applique qu'aux ensembles finis. On obtient une notion plus générale en observant qu'un élément du produit de E devrait "se composer" d'un élément quelconque de chacun des éléments de E. Il faut naturellement que l'élément particulier de E dont provient un composant du produit soit identifiable. L'ensemble ΠE contiendra donc les fonctions qui, à chaque élément x de E, associent un élément de x.

On aboutit alors à la définition générale suivante : si I est un ensemble et si, pour tout $i \in I$, A_i est un ensemble, on notera $\prod_{i \in I} A_i$, l'ensemble dont les éléments sont les fonctions de domaine I qui à tout élément $i \in I$ associent un élément $a_i \in A_i$ I, p. 61.

Définition 7.14. Un ensemble d'ensembles est un filtre défini sur l'ensemble I si :

$$\mathcal{F} \neq \varnothing \text{ et } \varnothing \notin \mathcal{F},$$

et si les trois conditions de clôture suivantes sont satisfaites :

1. $I \in \mathcal{F}$;
2. Si $X, Y \in \mathcal{F}$ alors $X \cap Y \in \mathcal{F}$;
3. Si $X \in \mathcal{F}$ et $X \subset Z \subset I$ alors $Z \in \mathcal{F}$.

Définition 7.15. Un ultrafiltre est un filtre qui, en plus des conditions 1-3, satisfait la condition suivante : "tout ensemble ou son complément est membre de $\mathcal{F}$", ce qui se note :

(7.2) $\quad X \in \mathcal{F} \vee (I/X) \in \mathcal{F}.$

Il ne faut pas confondre $\mathcal{F}$ au sens de filtre avec $\mathcal{F}$ au sens de structure. Le contexte suffit à lever l'ambiguïté.

Définition 7.16 : L'équivalence modulo un ultrafiltre [Blackburn et al.]. Soit I un ensemble non vide, soit U un ultrafiltre sur I. Considérons les ensembles non vides de A indexés par i. Soit le produit cartésien $\prod_{i \in I} A_i$ de tous ces ensembles. Comme on l'a vu dans la définition 7.13, le produit cartésien généralisé est un ensemble de fonctions. Ici, $\prod_{i \in I} A_i$ dénote l'ensemble de toutes les fonctions f avec domaine I telles que pour chaque $i \in I$, $f(i) \in A_i$.

Deux fonctions f et g peuvent être équivalentes modulo l'ultrafiltre.

Commençons par une caractérisation intuitive de ces conditions : Un filtre (et cela vaut aussi pour les ultrafiltres) sur I consiste dans les "gros" sous-ensembles de I et nous pouvons comprendre $f \backsim_F g$ (ici $f \backsim_U g$) comme disant que f s'accorde avec g sur presque toutes ses coordonnées [Bell and Slomson 69], p. 88.

Exemple. Soit $I = \{1, 2, 3\}$ et $E = \{A_1, A_2, A_3\}$, où $A_1 = \{4, 5\}, A_2 = \{6, 7\}, A_3 = \{8, 9\}$.

Le produit cartésien $\prod_{i \in I} A_i$ comporte 48 fonctions de choix représentées par autant de triplets. Les 8 triplets <4, 6, 8>, <4, 6, 9>, <4, 7, 8>, <4, 7, 9>, <4, 8, 6>, <4, 9, 6>, <4, 8, 7>, <4, 9, 7> sont les 8 fonctions qui s'accordent sur la première coordonnée de l'exemple considéré.

Formellement, la définition de l'équivalence modulo un ultrafiltre s'énonce

ainsi :

$$f \backsim_U g \Leftrightarrow \{i \in I / f(i) = g(i)\} \in U.$$

Définition 7.17. Sachant que la relation d'équivalence modulo U est une relation d'équivalence sur le produit cartésien $\prod_{i \in I} A_i$ qui est un ensemble de fonctions, on peut le diviser par $\backsim_U$ et obtenir des classes d'équivalence. L'ultraproduit de A_i est l'ensemble de toutes les classes d'équivalence de $\backsim_U$.

Définition 7.18. On peut définir l'ultraproduit non seulement pour des ensembles A_i mais aussi pour des structures $< A_i, R_i >$. Une collection de structures est dite *fermée sur les ultraproduits* si, lorsqu'on forme le produit de n'importe quelle sous-formule indexée de $C(\mathcal{F})$, elle est à son tour membre de $C(\mathcal{F})$.

7.3.6. *Théorème de Loš*

Théorème 7.6. Si une collection de structures $<A, R>$ peut être caractérisée par une formule du premier ordre, elle est fermée sur les ultraproduits.

Application à la logique modale

La traduction des formules de logique modale en formules de la logique du premier ordre (ou du deuxième ordre) permet d'utiliser le théorème de Loš pour distinguer les formules modales propositionnelles qui *correspondent* à des formules du premier ordre telles que $T, D, S4, S5$, etc... de celles qui *correspondent* à des formules du deuxième ordre tel que l'axiome de Mc Kinsey.

7.3.7. *Autres résultats de préservation*

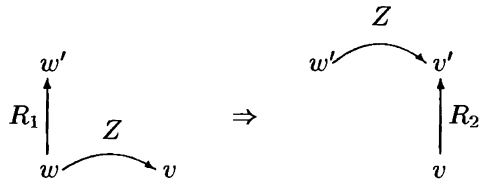
Définition 7.19. Les modèles $\mathcal{M}_1 = <W_1, R_1, V_1>$ et $\mathcal{M}_2 = <W_2, R_2, V_2>$ sont reliés par la relation de bisimulation Z si les conditions suivantes sont remplies [de Rijke 93], p. 42 :

(1) Si wZv, alors $\models_w p$ ssi $\models_v p$.

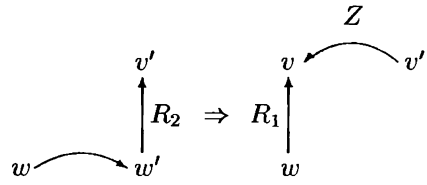
(2) Si wZv, $w' \in W_1$ et wR_1w', alors $w'Zv'$ pour quelque $v' \in W_2$ avec vR_2v' (condition dite "vers l'avant").

(3) Si wZv, $v' \in W_2$ et vR_2v', alors $w'Zv'$ pour quelque $w' \in W_1$ avec wR_1w' (condition dite "vers l'arrière").

La condition (2) peut se représenter ainsi :



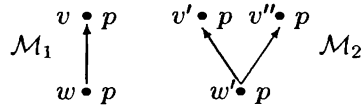
La condition (3) ainsi :



Concrètement, les conditions (2) et (3) requièrent qu'à chaque pas effectué dans un modèle corresponde un pas dans l'autre.

La relation de bisimulation entre modèles est *moins contraignante* que la relation d'isomorphisme. Par exemple, elle ne fait pas de distinction entre prolongation et ramification dans l'exemple ci-dessous.

Exemple.



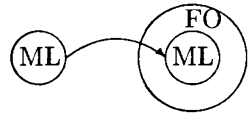
Une formule de la logique du premier ordre $P(x)$ est équivalente à sa traduction en logique modale propositionnelle si et seulement si elle est préservée sous les bisimulations.

Cet important résultat de M. de Rijke appelle trois commentaires :

(1) La bisimulation préserve les formules modales (les mêmes formules modales sont vraies dans les deux modèles bisimilaires $\mathcal{M}_1$ et $\mathcal{M}_2$).

(2) La bisimulation ne préserve pas les formules du premier ordre. Le modèle $\mathcal{M}_1$ vérifie $\forall_1 y \forall_2 y\,(xR_1 y \wedge xR_2 y \supset y_1 = y_2)$. Le modèle $\mathcal{M}_2$ vérifie la négation de cette formule, à savoir $\exists_1 y \exists_2 y\,(xR_1 y \wedge xR_2 y \wedge y_1 \neq y_2)$.

On voit ici que la traduction en logique du premier ordre *immerge* la logique modale (ML) dans la logique du premier ordre (FO).



La traduction dans l'autre direction se solderait par une perte d'information.

(3) Plusieurs manipulations de modèles laissent intacte la classe des formules modales que ces modèles rendent vraies. C'est le cas des *sous-modèles engendrés* dont il a été question antérieurement. En d'autres termes, elles produisent à partir d'un modèle $\mathcal{M}$ un modèle $\mathcal{M}'$ modalement équivalent. Plusieurs de ces constructions apparaissent comme autant de *cas particuliers de bisimulation.*

Chapitre 8
Théorie de la preuve pour la logique modale propositionnelle

8.1. Introduction

Dans les chapitres 2 et 3, la preuve par axiomes a été exposée et illustrée. D'autres méthodes de preuve ont également été élaborées pour la logique modale. On étudiera d'abord la méthode de déduction naturelle. La première présentation, à notre connaissance, est celle de Fitch [Fitch 52]. Anderson et Jonsthone exposent cette méthode de preuve pour le système $S4$ propositionnel [Anderson and Jonsthone 62] et Forbes le fait pour les systèmes $S5$ propositionnel et $S5$ quantifié [Forbes 94]. Nous adopterons la méthode de Fitting [Fitting 83] qui a l'avantage d'être modulaire : de légères modifications permettent d'obtenir les règles des logiques modales T, D, $S4$, $S5$ à partir de celles de K.

Après avoir formulé les règles de déduction naturelle pour les logiques modales K, T et $S4$, nous les illustrerons par des exemples. A titre d'exercice, le lecteur est invité à construire sur le même canevas la preuve des théorèmes de ces systèmes démontrés axiomatiquement au paragraphe 2.2.8.

Nous exposerons ensuite la méthode de preuve par "tableaux explicites" de Fitting [Fitting 83] dans la version de Nerode et Shore [Nerode and Shore 93]. Comme dans le tome I, on appliquera les différentes méthodes de preuve aux mêmes formules afin de faciliter la comparaison.

Viendra ensuite une étude détaillée de la méthode des tableaux implicites de R. Goré. Réclamant un peu plus d'inventivité de la part de l'utilisateur que la méthode des tableaux explicites, cette méthode a le mérite de la robustesse. Elle s'applique non seulement aux quinze systèmes de logique modale classique mais aussi à quelques autres tels ceux de la logique de la prouvabilité.

La méthode des tableaux implicites emprunte certains éléments à la méthode des séquents pour la logique modale.

Les preuves de correction et de complétude de la méthode des tableaux implicites pour le système $S4$ seront exposées. La preuve de complétude de Rautenberg et Goré sera présentée. Elle a l'avantage – par rapport à la preuve de complétude utilisant les modèles canoniques – d'être constructive et de déboucher sur une méthode de décision (pour les quinze systèmes normaux).

Cependant, si l'on s'en tient aux systèmes $K, T, D, B, S4$ et $S5$, la méthode des diagrammes de Hughes et Cresswell [Hughes and Cresswell 96] reste la plus maniable. Nous l'exposerons et l'illustrerons pour les systèmes K, T et $S4$.

8.2. Méthode de déduction naturelle

8.2.1. *Les règles et notions nouvelles*

La méthode de déduction naturelle en logique modale s'appuie sur la méthode de déduction naturelle en logique propositionnelle classique. Comme cette dernière, elle utilise la preuve par *hypothèse auxiliaire* et *par l'absurde*. Elle y ajoute des règles propres aux dérivations subordonnées.

Aux règles usuelles d'introduction et d'élimination des connecteurs en déduction naturelle (I, p. 134-153), il est commode d'ajouter la constante $\bot$:

$$\frac{A \quad \neg A}{\bot}$$

En logique classique, on fait usage de *dérivations subordonnées* introduites par des hypothèses auxiliaires. En logique modale, on y ajoute des *dérivations strictement subordonnées* introduites par des formules de la forme $\Diamond A$ ou $\neg\Box A$.

Nous désignerons par π les formules de la forme $\Diamond A$ et $\neg\Box A$ et par ν les formules de la forme $\Box A$ et $\neg\Diamond A$. Les instances des formules π et ν obtenues en éliminant l'opérateur modal de tête sont respectivement désignées par π_0 et ν_0.

Contrairement à la ligne initiale d'une dérivation subordonnée ordinaire qui est une hypothèse auxiliaire (une "supposition"), la ligne initiale d'une dérivation stricte nous fait passer d'un monde w où π était vrai à un monde w' accessible à partir de w où π_0 est vrai. Cette ligne, qui est de type π_0, est séparée de ce qui précède par une double ligne (voir figures 8.4 et 8.5).

Pour clore une dérivation subordonnée, on décharge l'hypothèse auxiliaire en l'incorporant comme antécédent d'un conditionnel qui a pour conséquent la dernière ligne de la dérivation subordonnée (voir figure 8.1).

Pour clore une dérivation subordonnée stricte dont la dernière ligne est le symbole de contradiction $\bot$, on indique par une double ligne qu'elle se termine et on répète le symbole $\bot$ (voir figure 8.2).

Aucune règle de décharge d'hypothèse auxiliaire (telle que la règle d'introduction de $\supset$) ne peut être appliquée à une formule figurant dans une dérivation subordonnée stricte, "pour l'évidente raison qu'il n'y a pas [dans cette situation] de supposition à décharger" [Fitting 83], p. 172. Remarquons que cette interdiction en présuppose une autre : on ne peut introduire de prémisse auxiliaire à l'intérieur d'une dérivation subordonnée stricte.

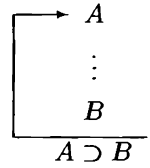


Figure 8.1

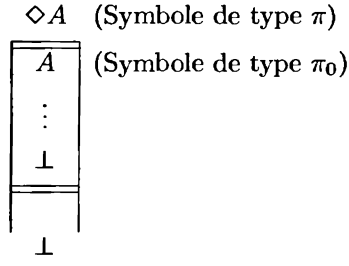


Figure 8.2

L'itération simple. Les dérivations dérivées peuvent être emboîtées. Toute formule de la dérivation qui a la plus grande portée peut être répétée dans une formule qui a une plus petite portée. C'est le cas de A dans la figure 8.3.

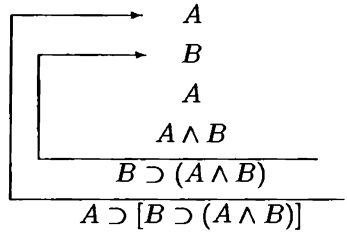


Figure 8.3

L'itération stricte. Appelons "boîte stricte" le rectangle formé en reliant deux doubles lignes latérales. Convenons de dire que l'apparition d'une formule

est au même niveau qu'une boîte si l'une et l'autre figurent à l'intérieur des mêmes boîtes. Convenons enfin de dire que l'apparition d'une formule est antérieure à une boîte dans une preuve si elle est au-dessus de la boîte.

Si **B** est une boîte (stricte) et si S est l'ensemble des formules qui font une apparition au-dessus de **B** et sont au même niveau d'emboîtement que **B**, alors une formule peut être répétée à l'intérieur de la boîte si elle est membre de l'ensemble $S\#$ défini ci-dessous pour les logiques K, $K4$, T, $S4$ et D [Fitting 93b]. Dans les exemples (figures 8.4 et 8.5), les formules $\Diamond A$ et $\Box B$ sont au même niveau d'emboîtement que la boîte **B**.

$$
\begin{aligned}
K\ S\# &= \{\nu_0 \mid \nu \in S\} \\
K4\ S\# &= \{\nu_0 \mid \nu \in S\} \cup \{\nu \mid \nu \in S\} \\
T\ S\# &= \{\nu_0 \mid \nu \in S\} \\
S4\ S\# &= \{\nu \mid \nu \in S\} \\
D\ S\# &= \{\nu_0 \mid \nu \in S\}
\end{aligned}
$$

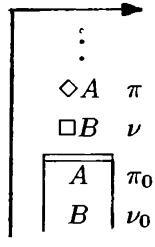


Figure 8.4

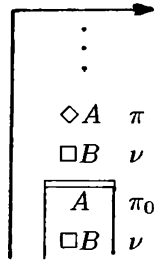


Figure 8.5

8.2.2. *Exemples de démonstration en déduction naturelle*

Démonstration dans K :

$$\vdash_K (\Box p \wedge \Diamond q) \supset \Diamond(p \vee q).$$

Utilisons $\neg\Box\neg$ au lieu de $\Diamond$:

1	$\Box p \wedge \neg\Box\neg q$	antécédent du théorème à démontrer
2	$\Box p$	de 1 par $\wedge$-él.
3	$\neg\Box\neg q$	de 1 par $\wedge$-él.
4	$\neg\neg\Box\neg(p \vee q)$	négation du conséquent
5	$\Box p$	itération de 2
6	$\neg\Box\neg q$	itération de 3
7	$\Box\neg(p \vee q)$	de 4 par double nég.
8	$\neg\neg q$	de 6 par règle π
9	q	de 8 par D.N.
10	$\neg(p \vee q)$	itération stricte de 7
11	p	itération stricte de 5
12	$\neg p$	de 10 par $\neg(\vee)$-él.
13	$\neg q$	de 10 par $\neg(\vee)$-él.
14	$\bot$	9,13,11,12
15	$\bot$	retour de 12
16	$\neg\neg\neg\Box\neg(p \vee q)$	preuve par l'absurde
17	$\neg\Box\neg(p \vee q)$	D.N.
18	$(\Box p \wedge \neg\Box\neg q) \supset \neg\Box\neg(p \vee q)$	décharge

Démonstration dans T :

$$\vdash_T \Box p \supset \Box\Diamond p.$$

1	$\Box p$	antécédent
2	$\neg\Box\neg\Box\neg p$	négation du conséquent
3	$\Box p$	réitération
4	$\neg\neg\Box\neg p$	de 2 par π
5	$\Box\neg p$	de 4 par D.N.
6	$\Box p$	de 3 par réitération stricte
7	$\neg p$	de 5 par règle ν pour T
8	p	de 6 par règle ν pour T
9	$\bot$	de 7 et 8
10	$\bot$	retour de 10
11	$\neg\neg\Box\neg\Box\neg p$	preuve par l'absurde (2-10)
12	$\Box\neg\Box\neg p$	de 11 par D.N.
13	$\Box p \supset \Box\underbrace{\neg\Box\neg p}_{\Diamond}$	

Démonstration dans $S4$:

$$\vdash_{S4} \Box(p \supset q) \supset \Box(\Box p \supset \Box q).$$

1	$\Box(p \supset q)$	
2	$\neg\Box(\Box p \supset \Box q)$	
3	$\Box(p \supset q)$	réitération de 1
4	$\neg(\Box p \supset \Box q)$	de 2 par π
5	$\Box p$	de 4 par $\neg(\supset)$-él.
6	$\neg\Box q$	de 4 par $\neg(\supset)$-él.
7	$\Box(p \supset q)$	itération stricte de 3 ($S\#$ pour $S4$)
8	$\neg q$	de 6 par π;
9	$\Box(p \supset q)$	itération stricte de 3 ($S\#$ pour $S4$)
10	$\Box p$	itération stricte de 5 ($S\#$ pour $S4$)
11	$p \supset q$	de 9 par règle ν pour $S4$
12	p	de 10 par règle ν pour $S4$
13	q	de 11 et 12 par $(\supset)$-él.
14	$\bot$	8, 13
15	$\bot$	retour
16	$\bot$	retour
17	$\neg\neg\Box(\Box p \supset \Box q)$	par l'absurde (4,15)
18	$\Box(\Box p \supset \Box q)$	de 16 par D.N.
19	$\Box(p \supset q) \supset \Box(\Box p \supset \Box q)$	décharge de 1

Remarque. La règle ν est commune à T et à $S4$ mais le type d'itération stricte utilisée en 9 et 10 est propre à $S4$. La démonstration de

$$\vdash_{S4} \Box p \supset \Box\Diamond\Box\Diamond p.$$

est laissée au lecteur.

Remarque. Tous les systèmes modaux comportent la règle $\frac{\pi}{\pi_0}$, c'est-à-dire $\frac{\Diamond A}{A}$ et $\frac{\neg\Box A}{\neg A}$.

En déduction naturelle, la différence entre les systèmes $K, T, K4, S4, D$ se manifeste :

(1) par les différentes formes que prend l'itération stricte (en K, on ne répète pas $\Box A$ mais seulement A. En $S4$, on répète $\Box A$) ;

(2) par des règles particulières propres au système :

(a) K et $K4$: aucune règle.

(b) T et $S4$ comportent la règle $\frac{\nu}{\nu_0}$, c'est-à-dire $\frac{\Box A}{A}$ et $\frac{\neg\Diamond A}{\neg A}$.

(c) D et $D4$ comportent la règle $\frac{\Box A}{\Diamond A}$.

L'article de A. Indrzejczak présente une vue d'ensemble sur les systèmes de déduction naturelle en logique modale propositionnelle [Indrzejczak 99].

8.3. La méthode des tableaux explicites

8.3.1. *Définition*

On appelle "méthode des tableaux explicites" une procédure de preuve par tableaux dans laquelle des informations relatives à la métathéorie tels que des énoncés sur la relation d'accessibilité et la vérité dans les mondes possibles figurent avec les formules du langage-objet dans le corps des dérivations. Cette méthode hybride est très intuitive, la justification sémantique des règles d'inférence figurant explicitement dans la formulation de celles-ci. Elle a été inventée par Fitting [Fitting 83]. Nous utiliserons dans un premier temps la version simplifiée de Nerode et Shore [Nerode and Shore 93].

8.3.2. *La preuve par tableaux explicites*

Pour prouver qu'une formule A est valide dans une classe de structures $C(\mathcal{F})$, ce qui se note $C(\mathcal{F}) \models A$, on construit un arbre de preuve. A la racine de celui-ci, on écrit l'énoncé : "il est faux, dans le monde d'origine w_0, que A", ce qui se note :

$$F_{w_0}(A).$$

On démembre ensuite la formule A en appliquant les *règles d'élimination* (appelées aussi *règles de réduction*) pour $\wedge, \vee, \supset, \neg$ (I, p. 174), auxquelles on ajoute les règles d'élimination des opérateurs modaux $\Diamond$ et $\Box$.

8.3.3. *Les règles d'élimination des opérateurs modaux*

La règle d'élimination de $\Diamond$ stipule que de "Il est vrai dans le monde w_i que possiblement A", on peut déduire "Il existe un monde possible w_j accessible à partir de w_i dans lequel il est vrai que A", ce qui se note :

$$\begin{array}{c} T_{w_i}(\Diamond A) \\ \downarrow \\ w_i R w_j \\ \downarrow \\ T_{w_j}(A) \end{array}$$

Ce monde doit être nouveau en ce sens que la constante w_j qui le désigne ne doit pas avoir fait d'apparition antérieure dans le tableau.

La règle d'élimination de $\Box$ stipule que de "Il est vrai dans w_i que nécessairement A", on peut déduire que "Pour tout monde w_j approprié, il est vrai que A". Ce qui se note :

$$T_{w_i}(\Box A)$$
$$\downarrow$$
$$T_{w_j}(A)$$

pour tout j approprié.

Le monde w_j est *approprié* s'il est apparu plus haut dans la branche.

Si ce n'était pas le cas, en K on serait bloqué. Par exemple, de $T_{w_0}(\Box p)$ on ne peut rien déduire tant qu'on a pas un monde w_k accessible à partir de w_0 et figurant sur la branche. Au contraire, à partir de $T_{w_0}(\Diamond p)$, on peut engendrer la transition $w_0 R w_1$ et le monde w_1 en appliquant la règle d'élimination de $\Diamond$ et inférer $T_{w_1}(p)$. Dans le système T, on n'aurait pas été bloqué. Le lecteur est invité à justifier la différence entre les systèmes K et T sur ce point.

Dans la méthode des tableaux pour le calcul des prédicats, on devait commencer par instancier les formules $T_w(\exists x P x)$ pour ensuite réutiliser les constantes individuelles obtenues par instantiation en les prenant comme instances pour les formules de la forme $T_w(\exists x P x)$. Dans la méthode des tableaux pour les calculs modaux, il en va de même pour les règles d'élimination de $\Diamond$ et de $\Box$. On commence par éliminer les opérateurs $\Diamond$, ce qui engendre de nouveaux mondes qui serviront de mondes d'accès lors de l'élimination des opérateurs $\Box$. On "recycle" à présent les constantes de monde comme on recyclait les constantes d'individus dans le calcul des prédicats.

8.3.4. *Règles duales*

Dans le calcul des prédicats, la règle :

$$T_w(\forall x P x)$$
$$\downarrow$$
$$T_w(Pc)$$

pour n'importe quel c approprié, a comme duale la règle :

$$F_w(\exists x P x)$$
$$\downarrow$$
$$F_w(Pc)$$

pour n'importe quel c approprié. Ces deux règles sont des règles γ.

La règle :

$$T_w(\exists x P x)$$
$$\downarrow$$
$$T_w(Pc)$$

pour un c nouveau, a comme duale la règle :

$$F_w \models \forall x P x$$
$$\downarrow$$
$$F_w \models Pc$$

pour un nouveau c. Ce sont des règles δ.

Dans la logique modale propositionnelle, la règle :

$$T_{w_i}(\Box A)$$
$$\downarrow$$
$$T_{w_j}(A)$$

pour tout monde w_j approprié, a comme duale la règle :

$$F_{w_i}(\Diamond A)$$
$$\downarrow$$
$$F_{w_j}(A)$$

Ce sont des règles ν. Et la règle :

$$T_{w_i}(\Diamond A)$$
$$\downarrow$$
$$w_i R w_j$$
$$\downarrow$$
$$T_{w_j}(A)$$

pour un w_j nouveau, a comme duale la règle :

$$F_{w_i}(\Box A)$$
$$\downarrow$$
$$w_i R w_j$$
$$\downarrow$$
$$F_{w_j}(A)$$

pour un w_j nouveau. Ce sont des règles π.

Les prédicats "appropriés" appliqués à une constante de monde w_j dans un tableau modal et à une constante d'individu dans la logique du premier ordre n'ont pas exactement le même sens. Le lecteur est invité à relever les différences.

8.3.5. *Tableau récapitulatif des règles*

8.3.5.1. *Règle α (de prolongation)*

(α)	$T_w(A \wedge B)$	$T_w(\neg\neg A)$	$F_w(A \vee B)$	$F_w(A \supset B)$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$(\alpha.1)$	$T_w(A)$	$T_w(A)$	$F_w(A)$	$T_w(A)$
	$\downarrow$		$\downarrow$	$\downarrow$
$(\alpha.2)$	$T_w(B)$		$F_w(B)$	$F_w(B)$

(α)	$T_w(\neg A)$	$F_w(\neg A)$
	$\downarrow$	$\downarrow$
$(\alpha.1)$	$F_w(A)$	$T_w(A)$

8.3.5.2. *Règle β (de ramification)*

(β)	$F_w(A \wedge B)$	$T_w(A \vee B)$	$T_w(A \supset B)$	
	$\swarrow \quad \searrow$	$\swarrow \quad \searrow$	$\swarrow \quad \searrow$	
(β_1, β_2)	$F_w(A) \quad F_w(B)$	$T_w(A) \quad T_w(B)$	$F_w(A) \quad T_w(B)$	
(β)	$T_w(A \equiv B)$	$F_w(A \equiv B)$		
	$\swarrow \quad \searrow$	$\swarrow \quad \searrow$		
(β_1, β_2)	$T_w(A) \quad F_w(A)$	$F_w(A) \quad T_w(A)$		
	$\downarrow \quad \downarrow$	$\downarrow \quad \downarrow$		
	$T_w(B) \quad F_w(B)$	$T_w(B) \quad F_w(B)$		

8.3.5.3. *Règle π (de possibilité)*

$T_{w_i}(\Diamond A)$	$F_{w_i}(\Box A)$
$\downarrow$	$\downarrow$
$w_i R w_j$	$w_i R w_j$
$\downarrow$	$\downarrow$
$T_{w_j}(A)$	$F_{w_j}(A)$
pour un j nouveau	pour un j nouveau

8.3.5.4. *Règle ν (de nécessité)*

$T_{w_i}(\Box A)$	$F_{w_i}(\Diamond A)$
$\downarrow$	$\downarrow$
$T_{w_j}(A)$	$F_{w_j}(A)$
pour tout j approprié	pour tout j approprié

8.3.6. *Exemple de preuve dans K*

Démontrons $\models_K (\Box p \wedge \Diamond q) \supset \Diamond(p \wedge q)$.

1	$F_w\,[(\Box p \wedge \Diamond q) \supset \Diamond(p \wedge q)]$	
	$\downarrow$	
2	$T_w\,[\Box p \wedge \Diamond q]$	de 1 par élimination de $\supset$ ($\supset$-él.)
	$\downarrow$	
3	$F_w\,[\Diamond(p \wedge q)]$	de 1 par $\supset$-él.
	$\downarrow$	
4	$T_w\,[\Box p]$	de 2 par $\wedge$-él.
5	$T_w\,[\Diamond q]$	de 2 par $\wedge$-él.
	$\downarrow$	
6	wRv	v nouveau, de 5 par π
	$\downarrow$	
7	$T_v\,[q]$	de 5,6 par π
	$\downarrow$	
8	$T_v\,[p]$	de 4,6 par ν
	$\downarrow$	
9	$F_v\,[p \wedge q]$	de 3,6 par ν
	$\swarrow \quad \searrow$	
10	$F_v\,[p] \qquad F_v\,[q]$	de 9 par $\wedge$-él.
	$\downarrow \qquad \downarrow$	
11	$\bot \qquad \bot$	de 8,10 et 7,10

8.3.7. *Les règles qui diffèrent selon les systèmes modaux*

Le cas de blocage évoqué plus haut sera évité, même si l'on ne dispose que d'un monde, à condition que la relation d'accessibilité soit réflexive. Pour être autorisé à éliminer un opérateur de nécessité, il faut disposer d'un monde accessible. Si la relation d'accessibilité est réflexive, ce qui est le cas dans T mais pas dans K, le fragment de dérivation suivant est correct :

$$\begin{array}{c} T_{w_0}\,[\Box p] \\ \downarrow \\ w_0 R w_0 \\ \downarrow \\ T_{w_0}\,[p] \end{array}$$

Le système $S4$ (c'est-à-dire $K + T + 4$) construit pour démontrer toutes les formules valides dans les structures réflexives et transitives comporte les deux règles suivantes :

1. Si τ est un tableau, nous pouvons le prolonger en τ' en ajoutant wRw à l'extrémité de chaque branche de τ sur laquelle w apparaît (règle de

réflexivité). Nous pouvons alors, comme on vient de le voir, passer de w_0 à w_0.

2. Si $w_L R w_J$ et $w_J R w_K$ apparaissent sur la branche b d'un tableau τ, alors nous pouvons prolonger τ en τ' en écrivant $w_L R w_K$ à l'extrémité de b et nous pouvons passer directement de w_L à w_K (règle de transitivité).

8.3.8. *La part du choix*

Une formule A est un théorème si et seulement s'il existe au moins un tableau fini fermé commençant par $F_w(A)$. Il peut y avoir plusieurs tableaux fermés, certains plus courts que d'autres.

8.3.9. *Exemple de démonstration dans S4*

Appliquons la méthode de Nerode et Shore à l'exemple suivant :

1	$F_{w_0}[\Box p \supset \Box\Diamond\Box\Diamond p]$	
	$\downarrow$	
2	$T_{w_0}[\Box p]$	de 1
	$\downarrow$	
3	$F_{w_0}[\Box\Diamond\Box\Diamond p]$	de 1
	$\downarrow$	
4	$w_0 R w_1$	de 3
	$\downarrow$	
5	$F_{w_1}[\Diamond\Box\Diamond p]$	de 1,4
	$\downarrow$	
6	$w_1 R w_1$	par réflexivité de R
	$\downarrow$	
7	$F_{w_1}[\Box\Diamond p]$	de 5,6
	$\downarrow$	
8	$w_1 R w_2$	de 7
	$\downarrow$	
9	$F_{w_2}[\Diamond p]$	de 7,8
	$\downarrow$	
10	$w_2 R w_2$	par réflexivilité de R
	$\downarrow$	
11	$F_{w_2}[p]$	de 9,10
	$\downarrow$	
12	$w_0 R w_2$	de 4,5 par transitivité
	$\downarrow$	
13	$T_{w_2}[p]$	de 2,12
	$\downarrow$	
14	$\perp$	de 2,13

On fait appel aux règles spéciales, soit quand on est bloqué (lignes 5 et 9), soit en vue d'obtenir la contradiction recherchée (ligne 12).

8.3.10. *Tableaux avec prémisses locales et tableaux avec prémisses globales*

La méthode des tableaux n'est pas limitée à la démonstration de théorèmes de logique modale. On peut l'utiliser aussi pour établir qu'une formule est dérivable de prémisses soit *locales* (appartenant à l'ensemble U) soit *globales* (appartenant à l'ensemble S) [voir 6.5.2 et 6.5.5].

Deux règles doivent être ajoutées [Fitting and Mendelsohn 98], p. 56 :

- *La règle de supposition locale*

  Si Y est n'importe quel membre de S, alors $1Y$ peut être ajouté à l'extrémité de n'importe quelle branche ouverte.

- *La règle de supposition globale*

  Si Y est n'importe quel membre de S, alors σY peut être ajouté à l'extrémité de n'importe quelle branche ouverte sur laquelle σ apparaît en tant que préfixe. Dans la notation de Fitting, différente de celle de Nerode et Shore, les suites de chiffres représentent les mondes. La relation d'accessibilité reste implicite.

Exemple. Voici un exemple dans lequel on démontre $\Box\Box p \supset p$ dans K en faisant usage de l'axiome T à titre de prémisse globale.

(1)		$\neg(\Box\Box p \supset p)$	
(2)		$\Box\Box p$	de 1 par $\supset$-él.[1]
(3)		$\neg p$	de 1 par $\supset$-él.
(4)		$\Box p \supset p$	Prémisse globale
(5)	1	$\neg\Box p$ | $1p$	de 4 par $\supset$-él.
(6)		$\neg p$	
(7)		$\Box p \supset p$	Prémisse globale
(8)		$\neg\Box p$ | $1\,1p$	de 7 par $\supset$-él.
(9)		$\Box p$ | $\bot$	de 2 par ν
(10)	1 1 1	$\neg p$	
(11)	1 1 1	p	
		$\bot$	

[1] Nous appellerons $\supset$-él. la règle qui permet d'éliminer $(P \supset Q)$ ou sa négation.

Exercice. Démontrer $(\Box p \supset p) \supset (\Box\Box p \supset p)$.

8.4. La méthode des tableaux implicites

8.4.1. *Le problème de la décision dans le calcul des propositions et dans le calcul des prédicats non modaux*

Dans le tome I figure une présentation du calcul des séquents de Gentzen qui respecte le principe de la sous-formule, principe qui s'énonce comme suit : " ... chaque formule qui est membre d'un séquent quelque part dans la démonstration est un constituant (une sous-formule) dans la conclusion de la démonstration ... " (I, p. 159). Ce principe de la sous-formule est également respecté dans la méthode de preuve par tableaux sémantiques.

Tant dans le calcul propositionnel des séquents que dans la méthode des tableaux, le respect du principe de la sous-formule a une conséquence importante. Il transforme une procédure de preuve en un *algorithme* de décision. La recherche systématique d'un contre-exemple aboutit après un nombre fini de pas soit à la production *effective* d'un contre-exemple, soit à la démonstration qu'il n'en existe pas, toutes les voies de recherche se fermant sur des contradictions.

Quand on passe au calcul des prédicats du premier ordre - dont Church a démontré en 1936 qu'il n'était que semi-décidable - la situation change. Même une méthode de preuve par tableaux ou par séquents qui respecte le principe de la sous-formule, n'aboutit pas toujours à un arbre fini. Il y a en effet trois issues :

- On aboutit à un arbre fini fermé (chaque branche se termine par une contradiction de la forme $<Tp, Fp>$ dans la méthode des tableaux ou à un axiome de la forme $p \Rightarrow p$ dans le calcul des séquents).
- On aboutit à un arbre fini ouvert (qui engendre un modèle falsificateur fini).
- On aboutit à un arbre infini ouvert (qui donne naissance à un modèle falsificateur infini).

Qu'en est-il du calcul modal propositionnel ?

8.4.2. *Preuve constructive de la complétude*

R. Goré a construit une méthode de preuve par tableaux implicites qui couvre les 15 systèmes normaux de la logique modale et quelques autres. Cette méthode très générale admet une *preuve constructive* de complétude, c'est-à-dire une preuve de complétude dont on peut *extraire une méthode de décision.*

Cette extraction repose sur un théorème qui affirme que "si S est un système modal normal axiomatisable, alors si S a la propriété du modèle fini (p.m.f.), alors S est décidable". (L'exposé de § 8.4.2 à § 8.5 repose sur [Goré 92, 95].)

8.4.3. *La méthode des tableaux implicites*

A la différence de la méthode des tableaux explicites présentée en 8.3, la méthode des tableaux implicites ne fait pas explicitement référence à la relation d'accessibilité. La différence entre les preuves dans les quinze systèmes se marque implicitement par des différences dans les règles d'élimination des opérateurs $\Box$ et $\Diamond$.

Comme toute preuve par tableau, les preuves de Goré sont des preuves par réfutation. On suppose fausse la formule à démontrer et on élimine progressivement les opérateurs modaux et les connecteurs le long de chaque branche issue de la racine.

8.4.4. *La forme des règles dans la méthode des tableaux implicites*

Les règles ont la forme suivante, N et D servant d'abréviation pour "Numérateur" et "Dénominateur".

$$\frac{N}{D_1 \mid D_2 \mid \ldots \mid D_K}$$

Chaque règle se lit de haut en bas et signifie "si le numérateur est sémantiquement consistant(en anglais *satisfiable*), le dénominateur est sémantiquement consistant".

Comme dans le calcul des séquents, le numérateur et le dénominateur sont constitués par des ensembles de formules, parmi lesquelles peuvent aussi figurer des singletons.

Dans le numérateur, on trouve une formule appelée *formule principale*, construite à partir de l'alphabet $\{P, Q\}$ ainsi qu'un ensemble de variables représentées par la variable X. Les formules représentées par cet ensemble sont appelées *formules paramétriques*.

Dans le numérateur formé de k branches, on trouve des formules que nous appellerons *secondaires* (en anglais *side-formulae*), qui résultent de l'application d'une règle du calcul à la formule principale du numérateur et, à nouveau, la variable X.

Exemple. Dans $\frac{X \cup \{P \wedge Q\}}{X \cup \{P\} \cup \{Q\}}$, la variable X désigne un ensemble. $P \wedge Q$ est une formule principale du numérateur. P et Q sont des formules secondaires dans le dénominateur. Pour alléger la notation, on remplace $\cup$ par ; et on supprime les accolades, ce qui donne $\frac{X;P \wedge Q}{X;P;Q}$.

A chaque connecteur et à chaque opérateur correspond une règle d'élimination comme dans toute preuve par tableau. Pour réduire le nombre de règles, on se borne à formuler une règle pour les seuls connecteurs et opérateurs primitifs.

8.4.5. *Les règles structurelles*

Le calcul des séquents comporte, outre les règles opératoires, des règles structurelles telles que la règle d'échange, la règle de contraction et la règle d'affaiblissement. La méthode de preuve par tableaux exposée ici n'en comporte qu'une, la règle d'affaiblissement Θ, qui permet de convertir un tableau pour un ensemble donné Y en un tableau pour un ensemble plus grand $(X;Y)$ simplement en ajoutant $(X;Y)$ à la racine à titre de nouveau nœud [Goré 95], p. 17.

$$\Theta \qquad \frac{X;Y}{X}$$

La règle d'affaiblissement correspond au principe de monotonie qui affirme que l'adjonction de prémisses supplémentaires en logique classique préserve la conclusion initiale, principe que l'on peut représenter ainsi :

$$X \vdash B \Rightarrow X \cup Y \vdash B.$$

8.4.6. *La règle de coupure*

Le calcul des séquents comporte une *règle de coupure.* C'est la seule règle qui, au lieu d'éliminer quelque chose (un connecteur ou un opérateur), introduit une formule nouvelle dans le corps de la dérivation. On voit que cette règle fait perdre au calcul qui la contient le *principe de la sous-formule*, à moins que l'on puisse l'éliminer, ce que Gentzen a réussi à faire pour la logique intuitionniste, réussissant du même coup à prouver sa décidabilité.

La méthode des tableaux de Goré comporte une règle qui est le pendant, pour la méthode des tableaux implicites, de la règle de coupure pour le calcul des séquents. Elle s'énonce ainsi :

$$\frac{X}{X;p \mid X;\neg p}$$

C'est le tiers exclu. Elle introduit un élément étranger $p, \neg p$, qui fait perdre à la dérivation la propriété de sous-formule.

8.4.7. *Définitions préliminaires*

Définition 8.1. Rappelons la définition, déjà donnée en 7.1, de l'ensemble des sous-formules d'une formule A.

(1) $Sf(p) = \{p\}$ où p est une formule atomique;

(2) $Sf(\neg A) = \{\neg A\} \cup Sf(A)$;

(3) $Sf(A \wedge B) = \{A \wedge B\} \cup Sf(A) \cup Sf(B)$;

(4) $Sf(\Box A) = \{\Box A\} \cup Sf(A)$.

Réduite à ces quatre clauses, la définition s'applique aux seules formules en notation primitive, c'est-à-dire aux formules qui ne contiennent pas les symboles $\vee$ et $\Diamond$.

Définition 8.2. R. Goré étend cette définition aux ensembles finis de formules. Soit X un ensemble de formules. L'ensemble $Sf(X)$ est constitué par toutes les sous-formules des formules membres de X. Formellement :

$$Sf(X) = \bigcup_{A \in X} Sf(X).$$

Définition 8.3. Une règle des tableaux a *la propriété de la sous-formule* si et seulement si chaque formule dans les dénominateurs est sous-formule de quelque formule dans le numérateur. Un système de preuves par tableau CL a la propriété de la sous-formule si et seulement si chaque règle de CL possède la propriété en question [Goré 95].

Définition 8.4. Rappelons maintenant la notion de *degré d'une formule*, toujours en suivant Goré.

(1) Une formule atomique a le degré 0.

(2) Préfixer une formule A d'une négation augmente le degré d'une formule d'une unité.

(3) Il en va de même pour A préfixé de $\Box$.

(4) Infixer une conjonction entre deux formules A et B donne naissance à une nouvelle formule dont le degré est celui du membre de la conjonction qui a le degré le plus élevé, augmenté d'une unité.

Définition 8.5. On définit ensuite le *degré modal* d'une formule.

(1) Le degré modal d'une proposition atomique est 0.

(2) Préfixer une négation à une formule A laisse intact le degré modal de cette formule.

(3) Préfixer une formule à l'aide d'un opérateur de nécessité accroît le degré modal de A d'une unité.

(4) Le degré modal d'une conjonction est celui du membre de la conjonction qui a le degré modal le plus élevé.

Définition 8.6. Le *degré d'un ensemble de formules* est le degré de la formule qui a le degré le plus élevé.

Définition 8.7. Le *degré modal d'un ensemble de formules* est le degré modal de la formule qui a le degré modal le plus élevé.

L'intérêt qu'il y a à imposer à une procédure de preuve le respect du principe de la sous-formule peut maintenant être précisé. Chaque règle de preuve

par tableaux qui obéit à ce principe fait décroître d'une unité le degré d'un des dénominateurs. Ce qui nous assure que le travail de démembrement ou d'analyse *se termine.*

8.4.8. *Relâchement du principe de la sous-formule*

Une règle d'inférence de la preuve par tableaux peut violer le principe de la sous-formule de trois manières :

(1) En introduisant dans le dénominateur des formules qui ne figurent pas dans le numérateur.

(2) En conservant dans le dénominateur, à côté de la formule réduite, une copie de la formule à réduire.

(3) En introduisant dans le dénominateur des formules d'un degré (tant modal que non modal) supérieur à celui des formules qui figurent dans le numérateur.

Le cas (1) est illustré par la règle d'élimination de l'opérateur de nécessité dans K :

$$(K)\ \frac{\Box X; \neg\Box P}{X; \neg P}.$$

Rappelons que $\Box X$ est une abréviation de $\{\Box P | P \in X\}$ et que $\neg\Box X$ est une abréviation de $\{\neg\Box P | P \in X\}$. La violation de type (1) du principe de la sous-formule ne met pas en danger la terminaison de l'algorithme de démembrement. En effet, elle n'augmente pas la profondeur de la formule.

Le cas (2) est illustré par la règle d'élimination de l'opérateur de nécessité dans T :

$$(T)\ \frac{X; \Box P}{X; \Box P; P}.$$

La violation de type (2) ne met pas davantage en danger la terminaison de l'algorithme. En effet, elle conserve la complexité mais ne l'accroît pas.

Le cas (3) est illustré par la règle B :

$$(B)\ \frac{X; \neg\Box P}{X; \neg\Box P; P | X; \neg\Box P; \neg P; \Box\neg\Box P}.$$

La violation de type (3) ne menace pas la terminaison si la méthode des tableaux CL (ici CLB) considérée possède la propriété de la *sur-formule analytique* définie dans la section qui suit [CL est l'abréviation de "calcul logique"].

8.4.9. *La propriété de la sur-formule analytique*

Si l'objectif visé est de trouver une méthode générale de preuve par tableaux qui soit non seulement une procédure de preuve des schémas valides mais

un algorithme de décision, il faut définir une propriété moins rigide et plus modulaire que la propriété de la sous-formule. C'est ce qu'a fait R. Goré en introduisant la notion de propriété de la sur-formule analytique.

Définition 8.8. Un système de preuve par tableaux possède la *propriété de la sur-formule analytique* si et seulement si à chaque ensemble fini de X nous pouvons assigner *a priori* un ensemble fini X^*_{CL} tel que X^*_{CL} contient toutes les formules qui peuvent faire une apparition dans *n'importe quel* tableau pour X.

Par exemple, pour le système K, l'ensemble $X^*_{CK} = Sf(X) \cup \neg Sf(X) \cup \{\bot\}$.

Muni de cette notion, on peut formuler le lemme suivant :

Lemme 8.1. Si une procédure de preuve par tableaux possède la propriété de la sur-formule analytique, alors si l'ensemble X de prémisses est un ensemble fini, on ne peut former qu'un nombre fini de tableaux à partir de cet ensemble.

Grâce à ce lemme, la procédure de preuve par tableaux devient une méthode de décision.

La preuve de ce lemme – sur laquelle nous reviendrons plus loin – repose sur le fait que l'ensemble des sous-ensembles d'un ensemble fini est lui-même fini.

8.4.10. *Restriction sur la coupure*

La méthode des tableaux pour B contient une règle de coupure, mais la présence de cette règle ne prive pas la méthode de preuve de son statut de méthode de décision. En effet, il s'agit d'une *règle de coupure analytique*; les formules nouvelles introduites dans les dénominateurs sont des sur-formules de formules présentes dans le numérateur, contrairement à la règle de coupure standard qui introduit dans le dénominateur une formule P (respectivement $\neg P$), absolument nouvelle.

Sur la notion de coupure analytique, on consultera aussi l'article de D'Agostino et Mondadori [D'Agostino and Mondadori 94].

8.4.11. *Règles de réduction*

Règles propositionnelles

$$(\wedge)\ \frac{X; P \wedge Q}{X; P; Q};$$

$$(\vee)\ \frac{X; \neg(P \wedge Q)}{X; \neg P | X; \neg Q};$$

$$(\bot)\ \frac{X; P; \neg P}{\bot};$$

$$(\neg)\ \frac{X;\neg\neg P}{X;P}.$$

Règle structurelle (d'affaiblissement)

$$\theta\ \frac{X;Y}{X}.$$

Règles modales de K, T, $K4$, $S4$

$$(K)\ \frac{\Box X;\neg\Box P}{X;\neg P}.$$

On justifie ainsi la règle K. Supposons que $\neg\Box p$, c'est-à-dire $\Diamond\neg p$, et $\Box X$ soient vrais dans un monde w. Alors en vertu de la signification de $\Diamond$, il existe un monde w' accessible à partir de w tel que $\neg p$ est vrai dans w'. Or, en vertu de la signification de $\Box$, les formules membres de X sont vraies dans tous les mondes qui sont accessibles à partir de w, donc elles sont vraies dans w' qui remplit cette condition.

$$(T)\ \frac{X;\Box P}{X;\Box P;P}.$$

On justifie ainsi la règle T : on sait, par définition de $\Box$, que si $\Box P$ est vrai dans un monde w_i, P est vrai dans tous les mondes w_j accessibles à partir de w_i. Comme la relation d'accessibilté dans la structure à laquelle T correspond est réflexive, il existe un monde accessible, donc on obtient P dans le dénominateur. Comme ce monde est le monde d'où l'on est parti, on conserve X et $\Box P$ dans le dénominateur.

$$(K4)\ \frac{\Box X;\neg\Box P}{X;\Box X;\neg P}.$$

On justifie ainsi la règle K4. La présence de X et de $\neg P$ dans le dénominateur repose sur les mêmes raisons que dans K. La présence de $\Box X$ dans le dénominateur est due à la transitivité de la relation d'accessibilité.

$$(S4)\ \frac{\Box X;\neg\Box P}{\Box X;\neg P}.$$

La justification est laissée au lecteur.

8.4.12. *Exemples de démonstration*

Démonstration en K

Démontrons la validité de $\Box\, p \wedge \Diamond\, p \supset \Diamond\, (p \wedge q)$.

Supposons faux le théorème à démontrer, ce qui en notation primitive s'écrit :

$$
\begin{array}{lll}
(1) & \{\Box\, p \wedge \neg\Box\neg\, q; \neg\neg\Box\neg\, (p \wedge q)\} & \\
\hline
(2) & \Box p \wedge \neg\Box\neg\, q\ ;\ \Box\neg\, (p \wedge q) & (\neg\neg) \\
\hline
(3) & \Box p\ ;\ \neg\Box\neg\, q\ ;\ \Box\neg\, (p \wedge q) & (\wedge) \\
\hline
(4) & p\ ;\ \neg\neg\, q\ ;\ \neg\, (p \wedge q) & (K) \\
\hline
(5) & p\ ;\ q\ ;\ \neg\, (p \wedge q) & (\neg\neg) \\
\hline
(6) & p\ ;\ q\ ;\ \neg p \quad | \quad p\ ;\ q\ ;\ \neg q & (\wedge) \\
\hline
(7) & \bot \quad | \quad \bot & (\bot)
\end{array}
$$

Démonstration en $S4$

Les règles de $S4$ sont les règles de réduction pour le calcul des propositions classiques auxquelles on ajoute la règle T et la règle $S4$.

Démontrons la validité de $\Box\ p \supset \Box\Diamond\Box\Diamond\ p$ en $S4$. Supposons faux le théorème à démontrer, ce qui en notation primitive s'écrit :

$$
\begin{array}{lll}
(1) & \{\Box p; \neg\Box\neg\Box\neg\Box\neg\Box\neg p\} & \\
\hline
(2) & \Box p; \neg\neg\Box\neg\Box\neg\Box\neg p & (\mathrm{S4}) \\
\hline
(3) & \Box p; \Box\neg\Box\neg\Box\neg p & (\neg\neg) \\
\hline
(4) & \Box p; \Box\neg\Box\neg\Box\neg p; \neg\Box\neg\Box\neg p & (\top) \\
\hline
(5) & \Box p; \neg\Box\neg\Box\neg p & (\theta) \\
\hline
(6) & \Box p; \neg\neg\Box\neg p & (\mathrm{S4}) \\
\hline
(7) & \Box p; \Box\neg p & (\neg\neg) \\
\hline
(8) & \Box p; p; \Box\neg p & (\top) \\
\hline
(9) & p; \Box\neg p & (\theta) \\
\hline
(10) & p; \Box\neg p; \neg p & (\top)
\end{array}
$$

8.5. Correction, complétude et décidabilité

8.5.1. *Caractère non déterministe de la procédure*

En vertu de la complétude qui sera prouvée plus loin, pour toute formule valide A, il existe au moins un tableau τ fini fermé qui commence par $F_w A$. En vertu du caractère constructif de la procédure, une exploration finie permet de trouver un tableau fini fermé s'il en existe un et dans le cas contraire de construire un contre-exemple falsifiant la formule.

La situation n'est cependant pas la même pour le calcul classique des propositions. Dans ce cas, s'il existe un tableau fini fermé pour une formule, on le trouve mécaniquement en appliquant les règles. Un choix mal avisé dans

l'application des règles risque seulement de retarder l'obtention de la preuve et d'allonger le tableau. En outre on peut se passer complètement des règles structurales.

Dans le cas de la logique modale, il en va autrement :

(1) L'utilisation de la règle structurale d'*affaiblissement* est indispensable.

(2) Il peut être obligatoire de *recopier* certaines formules modales.

(3) Un choix mal avisé de l'*ordre d'application des règles* peut avoir un effet pervers.

(4) L'utilisation de la règle de coupure analytique peut être indispensable.

Ces aléas de la procédure de preuve par tableaux n'empêchent pas celle-ci d'être une méthode de décision en vertu du fait qu'il n'y a qu'un nombre fini d'arbres à considérer.

8.5.2. *Preuve de la correction de R. Goré*

Théorème 8.2 : Théorème de correction. La logique L est *correcte pour la conséquence* si et seulement si "A est dérivable de l'ensemble Y de formules" implique "A est une conséquence logique de l'ensemble Y". Formellement : $Y \vdash A \Rightarrow Y \models A$ (il s'agit ici de la *conséquence locale*). S'agissant de la méthode des tableaux, on peut reformuler ainsi le théorème de correction : s'il existe un tableau fini fermé commençant par $Y \cup \neg A$ alors $Y \cup \neg A$ est sémantiquement inconsistant. Un tableau fini fermé, c'est un arbre renversé (la racine est placée au-dessus) tel que chaque branche est finie et se termine par la constante $\bot$.

Lemme 8.3. Pour toutes les règles, si tous les dénominateurs sont sémantiquement inconsistants, le numérateur est inconsistant.

Démonstration du lemme. La preuve du lemme est une preuve par cas. On considère successivement toutes les règles du calcul. Bornons-nous au cas de la règle $K4$. Prouvons la contraposée : si le numérateur N est sémantiquement consistant (*satisfiable*), au moins l'un des dénominateurs D_i est sémantiquement consistant.

Pour prouver que $K4$ est correct pour toutes les structures transitives, on raisonne comme suit : Soit $\mathcal{M} = <W, R, V>$ un modèle où la relation R est transitive. Supposons vraies en w_i les formules de numérateur. En vertu de la définition de $\Box$, il existe un monde w_j accessible à partir de w_i tel que $\models_{w_j} X; \neg P$. Comme R est transitive, pour tout w_k, si $w_j R w_k$, alors $\models_{w_k} X$, donc – toujours par définition de $\Box$ – $\models_{w_j} \Box X$. Rassemblant les deux résultats, on a donc au dénominateur $\models_{w_j} X; \Box X; \neg P$.

S'il n'y a pas de w_k tel que $w_j R w_k$, alors $\models_{w_j} \Box X$ trivialement. En effet $\models_{w_j} \Box X$ se définit $\forall w_j (w_j R w_k \supset \models_{w_k} X)$ et cette dernière implication est vraie parce que l'antécédent est faux. Donc, dans ce cas aussi, on a $\models_{w_j} X; \Box X; \neg P$ au dénominateur.

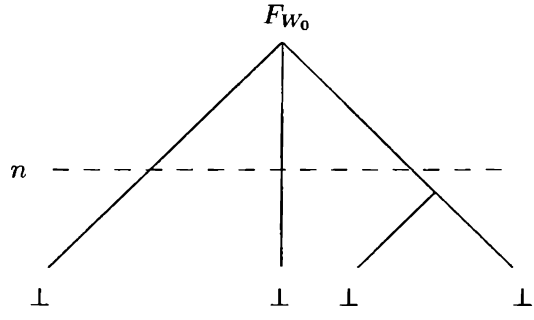
Remarque. La preuve de correction de la règle $K4$ donnée ici prouve également la correction de la règle $S4$.

Démonstration du théorème de correction. La preuve du *théorème de correction* procède par induction. Le cas de base est celui où une seule règle a été appliquée pour arriver à la fermeture. Ce ne peut être que la règle :

$$\frac{X; P; \neg P}{\bot}.$$

Dans ce cas, la formule qui décore la racine est bien sémantiquement inconsistante en raison de la définition de la valuation (ou plus simplement en raison de la table de vérité pour $P, \neg P$).

Considérons un arbre de longueur n. La longueur d'un arbre est la longueur de sa branche la plus longue. Le lemme sert, comme le note Goré, à faire "remonter l'inconsistance sémantique des feuilles terminales des branches jusqu'au sommet". On part des feuilles (c'est-à-dire de l'extrémité inférieure des branches), on remonte vers la racine le long des branches jusqu'à une certaine hauteur n



et on affirme que le théorème est vérifié aussi à la hauteur $n+1$. La justification de cette affirmation est immédiate en vertu du lemme ["Pour toutes les règles, si tous les dénominateurs sont sémantiquement inconsistants, le numérateur est sémantiquement inconsistant"].

8.5.3. *La notion d'ensemble saturé vers le bas*

Rappelons que l'objectif est d'élaborer une preuve de complétude constructive qui conduise à une méthode de décision. Comme on sait qu'un système qui a la propriété du modèle fini est décidable, on construit une preuve de complétude dans laquelle les modèles sont tous des modèles finis.

Dans la preuve de complétude par le modèle canonique, le modèle était basé sur une structure $\mathcal{F} = <W, R>$ dans laquelle les mondes possibles étaient des ensembles consistants maximaux de formules (une idée qui remonte à Bayart

[Bayart 59]). Pour former un *ensemble consistant maximal*, on part d'un ensemble de formules consistant et on le transforme en ensemble consistant maximal. Ce résultat peut aussi être obtenu par les deux opérations suivantes :

(1) On ajoute à l'ensemble certaines sous-formules : les sous-formules qui peuvent être ajoutées sans compromettre la consistance initiale. Par exemple, si l'ensemble consistant initial contenait la formule $\neg\neg A$, mais pas la sous-formule A, on ajoute cette dernière. Cette opération d'adjonction s'appelle la *saturation vers le bas.*

(2) On ajoute à l'ensemble certaines sur-formules : les sur-formules qui peuvent être ajoutées sans compromettre la consistance initiale. Par exemple, si l'ensemble consistant initial contenait A, mais pas $\neg\neg A$, on ajoutera cette dernière formule. Cette opération s'appelle la *saturation vers le haut.*

Les ensembles consistants maximaux sont saturés à la fois *vers le bas* et *vers le haut.* La définition récursive des ensembles maximaux consistants contiendra donc des clauses qui seront des *bi-implications*, par exemple la clause :

$$\neg\neg A \in X \Leftrightarrow A \in X.$$

Il y a une grande différence entre la saturation vers le bas et la saturation vers le haut. Si l'ensemble initial X était fini, la saturation vers le bas le sera aussi. Par exemple si $X = \{\neg\neg A, \neg\neg B, C\}$, l'ensemble X' saturé vers le bas sera $\{\neg\neg A, \neg\neg B, C, A, B\}$. Au contraire, la saturation vers le haut engendre des ensembles infinis. Par exemple si $X = \{A, B, C\}$, l'ensemble X'' saturé vers le haut sera l'ensemble infini $\{A, B, C, \neg\neg A, \neg\neg B, \neg\neg C, \neg\neg\neg\neg A, \neg\neg\neg\neg B, \neg\neg\neg\neg C, \ldots\}$.

En 1955, Hintikka a inventé une preuve constructive de complétude pour la logique classique du premier ordre qui n'utilisait que les ensembles saturés vers le bas, appelés aujourd'hui "ensembles de Hintikka". La preuve de complétude constructive de la logique modale $S4$ de Goré qui sera présentée ici fait usage de la notion d'ensemble saturé vers le bas.

Définition 8.9. Un ensemble X est dit *saturé vers le bas* s'il vérifie les clauses suivantes [Smullyan 68], [Goré 92] :

(1) Si $\neg A \in X$ alors $A \notin X$.

(2) Si $A \wedge B \in X$ alors $A \in X$ et $B \in X$.

(3) Si $A \vee B \in X$ alors $A \in X$ ou $B \in X$.

(4) Si $A \supset B \in X$ alors $\neg A \in X$ ou $B \in X$.

(5) Si $\neg\neg A \in X$ alors $A \in X$.

(6) Si $\neg(A \wedge B) \in X$ alors $\neg A \in X$ ou $\neg B \in X$.

(7) Si $\neg(A \vee B) \in X$ alors $\neg A \in X$ et $\neg B \in X$.

(8) Si $\neg(A \supset B) \in X$ alors $A \in X$ et $\neg B \in X$

On peut définir d'une manière plus concise les ensembles de Hintikka [Gallier 86], p. 90-91.

Définition 8.10 : La notion d'ensemble de Hintikka (ensemble saturé vers le bas). Un ensemble X de formules préfixées de V ou de F est un ensemble de Hintikka si et seulement s'il est fermé vers le bas pour les règles α (règles de prolongation) et pour les règles β (règles de ramification). Ou, ce qui revient au même, un ensemble X de formules préfixées par un symbole de polarité (V ou F) est un ensemble de Hintikka si et seulement si :

(1) Un atome et l'atome de polarité opposée ne figurent pas en même temps dans X.

(2) Si une formule α est dans X alors α_1 et α_2 sont dans X.

(3) Si une formule β est dans X alors β_1 est dans X ou β_2 est dans X.

8.5.4. *Vue d'ensemble sur la preuve de complétude de R. Goré*

Définition 8.11 : La complétude pour la conséquence pour $S4$. Si A est une conséquence logique de l'ensemble Y relativement à la classe des structures transitives et réflexives, alors A est dérivable de Y dans $S4$ ou plus précisément $Y; \neg A$ donne naissance à un tableau de $S4$ fini fermé.

Théorème 8.4. $S4$ est complet pour la conséquence.

Démonstration. On va prouver la contraposée, à savoir que si $Y; \neg A$ donne naissance à un tableau ouvert, $Y; \neg A$ est sémantiquement consistant. La démonstration de la contraposée comporte les étapes suivantes :

(1) On part de l'ensemble $Y; \neg A$, qui est tel que tous les arbres qu'on peut former en le démembrant sont des arbres ouverts. Soit τ un tel arbre. Etant ouvert, τ possède au moins une branche ouverte. Soit B une telle branche. La branche B étant ouverte, l'ensemble des formules qui décorent ses nœuds est un ensemble consistant. Appelons-le X.

(2) On étend X en un ensemble fini saturé vers le bas X' en appliquant les clauses de la définition récursive de "ensemble saturé vers le bas".

(3) On utilise à cette fin une procédure effective.

(4) On construit un modèle $\mathcal{M} = < W, R, V >$ dont les mondes sont des ensembles saturés vers le bas (le "graphe de modèle" de Rautenberg) et on montre que l'ensemble *syntaxiquement consistant* X' est *sémantiquement consistant*, c'est-à-dire que toutes les formules qu'il contient sont simultanément vraies dans le modèle.

(5) On montre que ce modèle est fini.

8.5.5. *La procédure de saturation vers le bas est effective*

L'ensemble X de départ est fini. Les règles qui transforment X en X' sont des opérations qui transforment un sous-ensemble de l'ensemble fini X^* en un autre sous-ensemble fini de X^* (voir la définition 8.8, § 8.4.9).

Pour une suite d'ensembles $X = X_0, X_1, \ldots$ obtenue par application successive des règles de réduction d'opérateurs propositionnels, la seule manière de ne pas se terminer serait d'amorcer un mouvement cyclique. Or, c'est impossible. La règle $(\bot)$ ne pourrait pas avoir été utilisée pour obtenir ce cycle. Les règles $(\wedge)$ et $(\top)$ ne peuvent pas non plus produire un cycle. En effet, elles accroissent la cardinalité de l'ensemble lors d'une réduction et aucune des autres règles ne la fait décroître. (On ne risque donc pas une oscillation périodique : accroissement-décroissement). Enfin, les règles $(\vee)$ et $(\neg)$ font décroître le degré de la formule [Goré 92], p. 56. Ajoutons que $S4$ fait décroître le degré modal.

8.5.6. *Construction du graphe de modèle*

Définition 8.12. Un *graphe de modèle* [le concept est dû à Rautenberg] pour un ensemble fini X de formules est une structure finie $<W_0, R>$ de L telle que :

(1) Les mondes $w \in W_0$ sont des ensembles saturés vers le bas de la logique L qui sont contenus dans X_L^*.

(2) La relation d'accessibilité satisfait les trois conditions suivantes :

(2.1) $X \subset w_0$ pour quelque $w_0 \in W_0$.

(2.2) Si $\neg\Box P \in w$ alors il existe un $w' \in W_0$ tel que wRw' et $\neg P \in w'$.

(2.3) Si wRw' et $\Box P \in w$, alors $P \in w'$.

Pour obtenir un modèle $\mathcal{M} = <W_0, R, V>$ à partir de la structure appelée ici graphe de modèle, il faut ajouter une valuation V à $\mathcal{F} = <W_0, R>$. Il est d'usage d'assigner comme valeur à un atome propositionnel la classe des mondes possibles dans lesquels il est vrai. Comme ici les mondes possibles sont des ensembles de formules, Rautenberg assigne comme valeur à un atome propositionnel la classe des ensembles de formules dont l'atome propositionnel est membre. Il convient alors de démontrer qu'il est possible d'étendre la valuation V pour les atomes propositionnels en valuation V' pour toutes les formules. En d'autres termes, on va devoir prouver que pour toute formule A, de complexité quelconque, si A est membre du monde w (qui est, en l'occurrence, un ensemble de formules saturé vers le bas), alors ce monde w rend vrai A, et si $\neg A$ est membre du monde w, alors w ne rend pas vrai A.

8.5.7. *Lemme d'existence d'un modèle*

Lemme 8.5. Si $<W_0, R>$ est une *structure* pour l'ensemble X, alors il existe un *modèle* pour X.

Démonstration [Goré 1992] La démonstration se fait par induction simultanée sur le degré de A. Il est présupposé que les ensembles saturés vers le bas sont fermés sur les règles de réduction de la logique considérée.

1. Cas de base

1.1. Soit A, l'atome propositionnel p. On doit démontrer que :

(a) $p \in w$ implique que $w \models p$.

Donnons-nous l'hypothèse, à savoir $p \in w$.
Or $V(p) = \{w \in W | p \in w\}$;
donc $w \in V(p)$.
Or par définition de $\models$ on a $\models_w p \Leftrightarrow w \in V(p)$;
donc $\models_w p$.

1.2. $\neg p \in w$ implique que $\not\models_w p$.
Comme w est un ensemble consistant, $p \notin w$;
donc $w \notin V(p)$;
donc $\not\models_w p$.

2. Induction

Considérons uniquement les clauses modales.

2.1 Soit $A = \Box B$.

Par la définition du graphe de modèle (Rautenberg), on a :

(1) $wRw' \wedge \Box P \in w \Rightarrow P \in w'$.

Par des transformations triviales du calcul des propositions, on passe de (1) à (2) :

(2) $\Box P \in w \Rightarrow (wRw' \Rightarrow P \in w')$;

(3) $P \in w' \Leftrightarrow \models_{w'} P$ par hypothèse inductive;

(4) $\Box P \in w \Rightarrow (wRw' \Rightarrow \models_{w'} P)$; par échange d'équivalents

(5) $\Box P \in w \Rightarrow \models_w \Box P$, par définition de $\models$ pour $\Box$.

2.2 Soit $A = \neg\Box A$.

Par la définition du graphe de modèle de Rautenberg, on a :

(1) $\neg\Box P \in w \Rightarrow \exists w' (wRw' \wedge \neg P \in w')$;

(2) $\neg P \in w' \Leftrightarrow w' \models_{w'} \neg P$ par hypothèse inductive;

(3) $\neg\Box P \in w \Rightarrow \exists w' (wRw' \wedge w' \models_{w'} \neg P)$;

(4) $\neg\Box P \in w \Rightarrow \models_{w'} \neg\Box P$, par définition de $\models$ pour $\neg\Box$.

Complétude de CS4. Partant de l'ensemble fini et sémantiquement consistant, on le sature vers le bas en appliquant la procédure effective pour obtenir

$X^* \subseteq X^*_{S4}$ (la saturation vers le bas préserve la consistance); cela conduit à un X^* fini si X était fini.

Si aucune formule de la forme $\neg\Box P$ ne figure dans X^*, le graphe de modèle sera $<\{w_0\}, \{w_0, w_0\}>$ [Goré 92], p. 60.

En effet, ce modèle satisfait les trois clauses définitoires du "graphe de modèle" de Rautenberg. En outre, il remplit les conditions pour être une structure $S4$ (réflexivité et transitivité).

Si X^* contient une ou plusieurs formules de la forme $\neg\Box P$, alors des mondes $w_1, w_2, \ldots$ sont engendrés à l'infini.

Cependant, comme les mondes sont des sous-ensembles de l'ensemble fini X^*, "une suite $w_0 < w_1 < \ldots$ dans W_0 doit, soit se terminer, soit être telle qu'un nœud se répète [Rautenberg 83], p. 413". Dans le deuxième cas, si $n > m$ sont minimaux avec $w_n = w_m$, continue Rautenberg, on arrête la construction et on identifie w_n, w_m dans W_0, obtenant de la sorte un *cercle* au lieu d'une *branche infinie.*

L'avantage de cette preuve de complétude, par rapport à la preuve par le modèle canonique, c'est de nous assurer non seulement que la logique est complète mais aussi qu'elle possède la propriété du modèle fini, et donc la décidabilité, sans qu'on ne doive utiliser la filtration.

8.6. La méthode des graphes cycliques orientés avec racine

L'introduction par R. Goré de règles structurales en plus des règles opérationnelles dans la méthode des tableaux et la distinction entre règles statiques (le numérateur N et le dénominateur D représentent le même monde) et règles de transition (N et D représentent des mondes différents) a considérablement assoupli la méthode des tableaux.

Une autre avancée doit être signalée [Castilho et al. 97] : la possibilité de remplacer les *tableaux* dans lesquels la progression se fait toujours de haut en bas par des *graphes cycliques orientés avec racine* qui autorisent d'autres modes de progression. Les tableaux, dans cette version nouvelle, cessent donc d'être des *arbres.* On se libère ainsi d'une contrainte qui rendait difficile la construction de méthodes de tableaux pour les logiques fondées sur les axiomes de densité ($\Diamond p \supset \Diamond\Diamond q$) ou de confluence ($\Diamond\Box p \supset \Box\Diamond p$). C'est là une lacune sérieuse de la méthode classique des tableaux, eu égard à l'importance intrinsèque de la propriété de confluence. On peut montrer, en effet, que la reproductivité, la réflexivité, la symétrie et la transitivité sont des *formes appauvries de confluence* [Popkorn 94], p. 65-66.

La méthode de preuve présentée par les quatre auteurs précités est une *généralisation* de la méthode des tableaux. Elle permet de construire des logiques qui expriment des propriétés de la relation d'accessibilité qui ne sont pas prises en compte par les méthodes classiques telles que la *quasi-réflexivité* ou la *quasi-transitivité.*

8.7. La méthode des diagrammes

8.7.1. *Le test de validité dans la logique classique*

On peut appliquer la méthode des tables de vérité de manière accélérée en considérant uniquement les cas susceptibles de rendre fausse la proposition testée. Par exemple, pour tester la validité de $p \supset (q \supset p)$, on supposera que la formule est fausse et on calculera les valeurs que doivent recevoir les sous-formules pour que cette supposition se vérifie. Dans l'exemple considéré, le test s'opère en trois étapes. Nous désignons **V** par 1 et **F** par 0.

(1) $p \supset (q \supset p)$;
0

(2) $p \supset (q \supset p)$;
$1\ 0 \quad 0$

(3) $p \supset (q \supset p)$.
$\underline{1}\ 0 \quad 1\ 0\ \underline{0}$

La dernière assignation est inconsistante.

Il arrive qu'il faille considérer des assignations alternatives :

(1) $p \supset (q \equiv r)$;
0

(2) $p \supset (q \equiv r)$;
$1\ 0 \quad 0$

(3) (i) $p \supset (q \equiv r)$ (ii) $p \supset (q \equiv r)$.
$1\ 0 \quad 1\ 0\ 0 \qquad 1\ 0 \quad 0\ 1\ 0$

On n'est pas arrivé à des valuations inconsistantes dans toutes les branches partant de la formule (1) *supposée fausse*. Donc, cette formule n'est pas valide. La valuation $V(p) = 1, V(q) = 1, V(r) = 0$ est un contre-exemple. De même la valuation $V(p) = 1, V(q) = 0, V(r) = 1$.

8.7.2. *Le test des diagrammes pour K*

Introduction. La méthode accélérée a été étendue par Hughes et Cresswell aux systèmes $K, T, D, S4, S5$.

Dans le test de validité pour le calcul des propositions, on travaillait uniquement sur des molécules construites à l'aide d'atomes propositionnels, de connecteurs et de parenthèses. Pour ajuster le test aux formules du calcul propositionnel modal, il faut étendre aux formules $\Box B$ et $\Diamond B$ la règle d'assignation de valeurs aux formules. Comme les opérateurs modaux n'expriment pas des *fonctions de vérité*, l'extension ne va pas de soi. La méthode des diagrammes intègre des composants de la méthode des tables mais aussi des composants de la méthode des tableaux.

Règles des astérisques. Si l'on cherche à falsifier A et si l'on doit, pour progresser vers ce but, assigner 1 à une formule de la forme $\Box B$ ou assigner 0 à une formule de la forme $\Diamond B$, alors on indique 1 (respectivement 0) sous l'opérateur modal et on écrit un astérisque *au-dessus*.

Si, au contraire, on doit, pour progresser vers le but visé, assigner 0 à une formule de la forme $\Box B$ ou assigner 1 à une formule de la forme $\Diamond B$, alors on indique 0 (respectivement 1) sous l'opérateur modal et on écrit un astérisque *en dessous*.

Par exemple, pour tenter de réfuter $(\Box p \wedge \Diamond q) \supset \Diamond(p \vee q)$ en K, on commence par faire les assignations (1)-(3) :

$$\begin{array}{lccccc}
(1) & (\Box p \wedge \Diamond q) & \supset & \Diamond & (p \vee q); \\
 & & 0 & & \\
(2) & (\Box p \wedge \Diamond q) & \supset & \overset{*}{\Diamond} & (p \vee q); \\
 & \quad 1 & 0 & 0 & \\
(3) & \overset{*}{(}\Box p \wedge \Diamond q) & \supset & \overset{*}{\Diamond} & (p \vee q). \\
 & 1\ 1\ \underset{*}{1} & & &
\end{array}$$

Règles relatives aux mondes

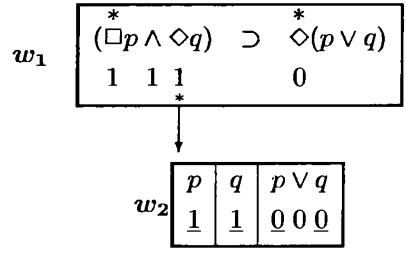
A. Si dans un monde w figure une formule de la forme $\Box B$ avec l'astérisque au-dessus ($\overset{*}{\Box} B$), alors pour tout monde w' accessible à partir de w, on doit assigner 1 à B si toutefois il existe un tel monde accessible.

B. Si dans un monde w figure une formule de la forme $\Diamond B$ avec l'astérisque au-dessus ($\overset{*}{\Diamond} B$), alors dans tout monde w' accessible à partir de w, on doit assigner 0 à B si toutefois il existe un tel monde accessible.

C. Si dans un monde w figure une formule de la forme $\Box B$ avec un astérisque au dessous ($\Box \overset{*}{B}$), alors il *doit* exister un monde w' accessible à partir de w, monde w' dans lequel on assignera la valeur 0 à B.

D. Si dans un monde w figure une formule de la forme $\Diamond B$ avec un astérisque au dessous ($\Diamond \overset{*}{B}$), alors il *doit* exister un monde w' accessible à partir de w, monde w' dans lequel on assignera la valeur 1 à B.

Exemple.



Le deuxième rectangle contient deux assignations inconsistantes : $V(p) = 1, V(p) = 0, V(q) = 1, V(q) = 0$.

Remarques. Les règles A et B peuvent se formuler ainsi :

A'. Si $V(\Box B) = 1$ alors pour tout w' si wRw' alors $V(B) = 1$.

B'. Si $V(\Diamond B) = 0$ alors pour tout w' si wRw' alors $V(B) = 0$.

On remarque que A et B contrairement à C et D n'affirment pas l'existence de mondes. La portée de cette remarque se voit bien quand on tente (vainement) de prouver $\Diamond(p \supset \Box q)$ en K.

Cette formule n'est pas un théorème de K. En effet, on ne dispose pas d'un monde accessible auquel on accéderait en éliminant l'opérateur $\Diamond$. Faute d'avoir un tel monde, on est bloqué. On ne peut appliquer la règle B.

Exemple.

w_1 | $\overset{*}{\Diamond}(p \supset \Box p)$
 0

Les règles C et D servent à engendrer de nouveaux mondes et rendent applicables les règles A et B. Ainsi, dans l'exemple 1, on commence par utiliser la règle D pour *créer* le monde w_2 et éliminer le $\Diamond$ dans $\Diamond q$. Cette opération ayant été exécutée, on peut appliquer la règle A à $\overset{*}{\Box} p$ et B à $\overset{*}{\Diamond} q$. Les règles C et D correspondent à la règle π dans la méthode des tableaux pour K.

8.7.3. *Le test des diagrammes pour T*

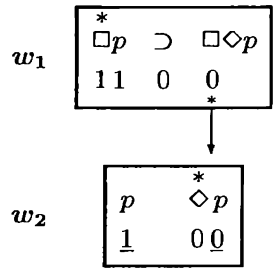
Quand on est dans une structure T, c'est-à-dire une structure réflexive, il ne faut pas attendre la production d'un monde obtenue par l'application des règles C et D pour avoir le droit d'employer l'une des règles A et B. On peut, en effet, utiliser le monde représenté par le rectangle sur lequel on travaille comme point de départ et aussi comme point d'arrivée de la transition wRw'.

Cette possibilité nouvelle offerte dans le système T se traduit par la modification suivante à la règle des astérisques : là où un astérisque apparaît au-dessus de $\Box$ ($\overset{*}{\Box}$), la formule qui suit doit recevoir la valeur 1. Là où un astérisque apparaît au-dessus de $\Diamond$ ($\overset{*}{\Diamond}$), la formule qui suit doit recevoir la valeur 0 [Hughes and Cresswell 96], p. 77.

Grâce à cette version nouvelle des règles A et B qui repose sur la possibilité d'utiliser le monde de départ comme monde d'arrivée, on peut prouver en T la formule $\Diamond(p \supset \Box q)$ qu'on ne pouvait prouver en K.

Démontrons à présent par la méthode des diagrammes un théorème de T déjà démontré par la méthode de la déduction naturelle et des tableaux.

Exemple.



Le lecteur est invité à dire pourquoi cette formule n'est pas un théorème de K.

8.7.4. *Le test des diagrammes pour S4*

Pour formuler la règle qu'il faut ajouter à K et T pour obtenir un test pour $S4$, on doit d'abord définir le notion de chaîne :

Définition 8.13. Dans un diagramme sémantique, une série de rectangles $w_1, \ldots, w_n$ forme une chaîne si une flèche part de chaque rectangle (le dernier exclu) vers le rectangle suivant dans la série [Hughes and Cresswell 96], p. 85.

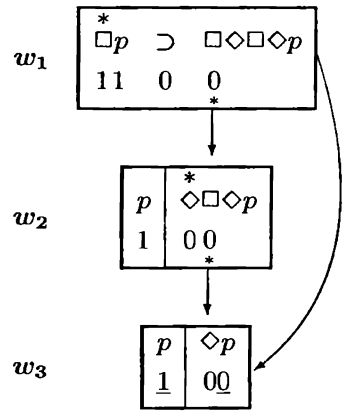
Notons que lorsqu'un rectangle se ramifie en rectangles alternatifs, chacun de ceux-ci appartient à la chaîne à laquelle appartient le rectangle d'où est issue la ramification.

Le test pour les diagrammes de $S4$ doit contenir une clause qui reflète la transitivité de la relation d'accessibilité.

On le formule ainsi : "Un diagramme de $S4$ diffère d'un diagramme pour T de la manière suivante : une flèche doit aller de chaque rectangle vers chaque autre rectangle qui apparaît plus loin dans chaque chaîne à laquelle le premier rectangle appartient " [Hughes and Cresswell 96], p. 85.

Les règles A et B sont modifiées ainsi en $S4$: quand un rectangle comporte la formule $\underset{1}{\Box}B$ (respectivement $\underset{0}{\Diamond}B$), il faut écrire $\underset{1}{B}$ (respectivement $\underset{0}{B}$) non seulement dans le rectangle suivant, comme c'était le cas en T mais aussi dans tout autre rectangle ultérieur (exemple suivant).

Exemple. Démonstration dans $S4$ avec usage (indispensable) des alternatives.



Le lecteur est invité à distinguer les assignations faites en vertu de T des assignations faites en vertu de $K4$. ($S4 = K + 4 + T$)

8.7.5. *Méthode de preuve par réfutation et méthode de décision*

La méthode des diagrammes pour tester la validité dans T constitue une *méthode de décision.* En effet, si la formule testée est un théorème, on arrive mécaniquement à un diagramme qui se termine sur des inconsistances et si la formule est un non-théorème, le diagramme nous fournit un modèle fini dans lequel la formule est fausse.

Pour T, le caractère fini du modèle falsificateur se déduit du caractère fini des diagrammes, lequel provient du fait qu'en T le nombre des opérateurs décroît au fur et à mesure qu'on progresse dans la construction du diagramme. D'où l'on aboutit inévitablement à un rectangle qui ne contient que des formules non modales et ces rectangles ne peuvent en engendrer de nouveaux [Hughes and Cresswell 96].

Cette *preuve de terminaison* très intuitive nous assure que la méthode de preuve pour T est aussi une méthode de décision.

Qu'en est-il pour $S4$? Si on applique le test de validité de $S4$ à la formule de Mc Kinsey $\Box\Diamond p \supset \Diamond\Box p$, on engendre un diagramme comportant deux branches infinies [Hughes and Cresswell 96], p. 87, 108. Cette formule n'est pas valide en $S4$ mais à première vue on ne peut construire un modèle fini qui la falsifie. Il y a cependant une solution au problème.

Comme la méthode des diagrammes respecte la propriété de la sous-formule, on peut prouver par le raisonnement suivant (qui rejoint celui de Goré et de Rautenberg) que ces branches infinies comportent des répétitions.

- Les règles de construction des diagrammes imposent que chaque formule apparaissant dans le diagramme soit une sous-formule de la formule testée.

- Cette formule n'a qu'un nombre fini de manières d'assigner des valeurs de vérité à ces sous-formules.
- Donc "dans une chaîne infinie, on arrivera tôt ou tard à rencontrer un rectangle qui est contenu dans un rectangle antérieur [Hughes and Cresswell 96], p. 90".

A ce stade, la solution consiste à réorienter la chaîne qui part du rectangle que nous appellerons "répétitif" vers le rectangle antérieur qui le contenait plutôt que vers un nouveau rectangle. Nous obtenons alors une chaîne finie qui se termine par une boucle. Toutes les chaînes traitées de la sorte sont finies.

Or il n'y a qu'un nombre fini de chaînes dans un diagramme de $S4$ puisque chaque rectangle ne contient qu'un nombre fini d'opérateurs modaux.

Tout diagramme de $S4$ ne contient donc qu'un nombre fini de rectangles.

La méthode des diagrammes de $S4$, dont la complétude est démontrée par les auteurs est donc aussi une méthode de *décision.*

8.7.6. *La méthode des tableaux pour la logique temporelle*

La procédure de preuve par tableaux présentée dans [Wolper 85] couvre la logique temporelle présentée au paragraphe 3.2.4. La procédure s'étend en outre aux formules contenant des occurrences de l'opérateur binaire $\mathcal{U}$. Elle est donc une extension conservatrice de la logique temporelle linéaire dont l'axiomatique est fournie en 3.2.4.2.

Le principe de la procédure de preuve développée dans l'article de 1985 se rattache à la famille des preuves par tableaux. A partir du singleton contenant la formule dont on veut tester la satisfaisabilité, on construit un graphe orienté dont les nœuds sont des ensembles de formules. On engendre de nouveaux nœuds à partir des nœuds existants en appliquant des règles.

Contrairement à ce qui a lieu dans la plupart des procédures de preuve par tableaux, toutes les règles ne se bornent pas à éliminer un opérateur. Certaines règles, au contraire, régénèrent indéfiniment l'opérateur sur lequel elles portent. Il s'ensuit qu'il arrive que deux nœuds situés sur un même chemin soient décorés de la même étiquette, c'est-à-dire soient étiquetés par le même ensemble de formules. On arrête alors ainsi le développement de la branche : on relie le nœud décoré de l'étiquette répétée au nœud décoré de l'étiquette apparaissant pour la première fois au moyen d'une flèche de retour. En d'autres termes, on crée une boucle.

On doit, à présent, préciser sous quelles conditions une telle boucle compte comme un cas de *fermeture* de la branche. La formulation de ces conditions requiert l'introduction préalable des concepts de *formule élémentaire*, de *formule marquée*, d'*état*, de *pré-état* et de *satisfaisabilité dans un pré-état.*

Définition 8.14. On appelle *formules élémentaires* les formules qui sont soit des littéraux, soit des formules dont $\bigcirc$ est le connecteur principal.

Définition 8.15. Toute formule à laquelle une règle a été appliquée est conservée dans les nœuds subséquents, mais on signale à l'aide d'un astérisque qu'elle n'est plus utilisable. On appelle ces formules *formules marquées.* Il y a cependant une exception. Lors de l'application de la règle d'élimination des $\bigcirc$, les formules marquées ne sont pas conservées.

Définition 8.16. On appelle *état* un nœud étiqueté par un ensemble de formules auquel n'appartiennent que des formules marquées ou des formules élémentaires.

Définition 8.17. On appelle *pré-état* le nœud initial et tout nœud qui est le descendant immédiat d'un état.

Définition 8.18. Une formule de la forme $\Diamond f_2$ ou $f_1 \mathcal{U} f_2$ est sémantiquement consistante (*satisfiable*) dans un pré-état si et seulement s'il existe un chemin dans le tableau qui conduit de ce pré-état à un nœud contenant la formule f_2.

Cette définition permet de déclarer insatisfaisable – et de traiter comme branche fermée – un chemin qui part de $\Diamond f_1$ ou de $f_1 \mathcal{U} f_2$ figurant dans une configuration C et qui revient, par une boucle, à la même configuration C. En effet, quand une telle boucle se produit, on repousse indéfiniment la réalisation de "possiblement f_2" ou de "f_1 jusqu'à f_2". Or une possibilité qui ne se réalise jamais équivaut à une impossibilité.

Règles d'élimination pour les formules non élémentaires

R1 $\dfrac{\neg\neg f}{\{\{f\}\}}$ R2 $\dfrac{\neg\bigcirc f}{\{\{\bigcirc\neg f\}\}}$

R3 $\dfrac{f_1 \wedge f_2}{\{\{f_1, f_2\}\}}$ R4 $\dfrac{\neg(f_1 \wedge f_2)}{\{\neg f_1\} \mid \{\neg f_2\}}$

R5 $\dfrac{\Box f}{\{\{f, \bigcirc\Box f\}\}}$ R6 $\dfrac{\neg\Box f}{\{\neg f\} \mid \{\neg\bigcirc\Box f\}}$

R7 $\dfrac{\Diamond f}{\{f\} \mid \{\bigcirc\Diamond f\}}$ R8 $\dfrac{\neg\Diamond f}{\{\{\neg f, \neg\bigcirc\Diamond f\}\}}$

R9 $\dfrac{f_1 \mathcal{U} f_2}{\{f_2\} \mid \{f_1, \bigcirc(f_1 \mathcal{U} f_2)\}}$ R10 $\dfrac{\neg(f_1 U f_2)}{\{\{\neg f_2, \neg f_1 \vee \neg\bigcirc(f_1 U f_2)\}\}}$

Règles d'élimination pour les formules élémentaires:

R11 Si un nœud contient seulement des formules élémentaires ou des formules marquées, on crée un descendant immédiat de ce nœud contenant uniquement les formules dont l'opérateur principal est $\bigcirc$ et auxquelles cet opérateur principal a été retiré. On efface donc les littéraux et les formules marquées. De nouveaux littéraux peuvent cependant apparaître lorsque la formule élémentaire sur laquelle on travaille est formée d'un

littéral préfixé par $\bigcirc$.

Règles d'élimination des nœuds :

R12 On efface tout nœud qui contient deux littéraux complémentaires.

R13 Si un nœud qui est un pré-état contient une formule de la forme $\Diamond f$ ou $f_1 \mathcal{U} f_2$ qui n'est pas satisfaisable (voir définition 5), on l'efface.

R14 Si tous les successeurs d'un nœud ont été éliminés, ce nœud est à son tour éliminé.

Théorème 8.6 : Théorème de complétude. Une formule de *PTL* est sémantiquement consistante (satisfaisable) si et seulement si le graphe généré par la procédure de décision décrite plus haut n'est pas éliminé. La preuve de complétude est présentée dans l'article [Wolper 85], p. 126-127.

8.7.7. *Incomplétude de la logique temporelle linéaire du premier ordre*

La logique temporelle linéaire propositionnelle est complète, la logique non modale du premier ordre l'est aussi. On s'attendrait à ce que la logique temporelle linéaire du premier ordre obtenue en combinant les deux logiques précitées le soit également. Ce n'est pas le cas.

R. Kuiper et W. Penczek, en adaptant une preuve de Szala (1987), montrent qu'il n'existe pas d'axiomatisation finie et complète de la logique temporelle linéaire du premier ordre. Si on enrichit le langage à l'aide des trois symboles de fonction suivants : *successeur de*, + et *, on prouve l'incomplétude de la logique "en définissant un ensemble de formules *FN* tel que l'ensemble de ses conséquences sémantiques ne peut être déduit, quel que soit l'ensemble fini d'axiomes et de règles d'inférence pour la logique linéaire du premier ordre que l'on se donne" ([De Swart 93], p. 240).

Chapitre 9

Théorie des modèles et théorie de la preuve pour la logique modale quantifiée

9.1. Introduction

Le premier système axiomatique de la logique modale quantifiée est dû à Ruth Barcan Marcus [Barcan 46]. Ce n'est que vers la fin des années cinquante que la sémantique des mondes possibles pour la logique modale quantifiée se développe. Bayart, Guillaume, Hintikka, Kanger et Kripke ont indépendamment les uns des autres jeté les bases de cette sémantique ([Bayart 58, 59], [Guillaume 58a, b, c], [Hintikka 57], [Kanger 57], [Kripke 59]). L'article de référence est *"Semantic Considerations on Modal Logic"* [Kripke 63].

Avant l'apparition de la sémantique des mondes possibles, on savait que les systèmes de Lewis étaient logiquement différents. Il y a des théorèmes qu'on peut prouver en $S4$ mais pas en $S5$. Faute d'une sémantique pour ces systèmes, on n'arrivait pas à interpréter leurs axiomes. La traduction des axiomes en langage naturel ne nous aide guère. Que peut bien signifier "il est nécessairement nécessaire que $2 + 2 = 4$" ?

La sémantique des mondes possibles dont on peut se servir pour interpréter la logique modale quantifiée est plus complexe que celle qui nous a servi en logique modale propositionnelle. Elle fait un usage essentiel de la notion de *domaine*.

Dans leur introduction à *Signification et Nécessité* de Carnap, F. Rivenc et Ph. de Rouilhan ont isolé trois concepts de monde possible, de complexité croissante, intervenant dans la sémantique de la logique modale quantifiée.

Dans la première acception de "monde possible", un *monde possible* **G** pour le langage L sera un couple <**D**,**F**>, où **D** (domaine de **G**) est un ensemble non vide (de cardinalité quelconque), et **F** une fonction qui associe à chaque

symbole de prédicat n-aire un ensemble de n-uplets d'éléments de **D** (extension du symbole de prédicat dans **G**) [Rivenc et de Rouillan 97], p. 31.

Avec l'aide de la notion de *monde possible* ainsi définie (notion qui correspond à ce qu'en théorie des modèles ordinaires on appelle "modèle"), on peut définir celle de modèle pour la logique du premier ordre *quantifiée* : "Un *modèle* est un couple <**G**,**K**>, où **G** est un monde possible et **K** l'ensemble de tous les mondes possibles de même domaine que **G**. Ce domaine commun est, par définition, le domaine du modèle" [Rivenc et de Rouilhan 97], p. 31.

On définit ensuite par induction la satisfaction d'une formule **B** dans un modèle <**G**,**K**> pour une assignation s de valeurs aux variables libres. La clause pour l'opérateur de nécessité est :

$$<G, K> \models_s \Box S \text{ ssi pour tout } H \in K, H \models_s S.$$

F. Rivenc et Ph. de Rouilhan font observer que cette clause interprète $\Box$ par une quantification sur tous les mondes d'un modèle et qu'elle correspond à la *nécessité* conçue comme ce qui vaut dans toutes les "descriptions d'états"[1].

Le deuxième concept de monde possible introduit par Kripke diffère de celui qui vient d'être décrit en ceci que "la notion de modèle et, corrélativement, l'interprétation de l'opérateur de nécessité ne font plus référence à *tous* les mondes d'un domaine donné, mais seulement à *une partie quelconque* non vide d'entre eux".

La notion de nécessité qui correspond à la validité définie au moyen de la nouvelle notion de modèle n'est plus la notion de *nécessité logique*. Les mondes possibles ne sont plus comme précédemment des configurations *quelconques* d'états de choses sur la base du langage et du domaine de ce modèle. Il est plus naturel, comme l'écrivent F. Rivenc et Ph. de Rouilhan, "de les interpréter comme représentant, du point de vue du monde distingué dans le modèle, de ce que ce monde aurait *vraiment ... pu* être" [Rivenc et de Rouilhan 97], p. 34. C'est donc bien d'une *nécessité métaphysique* qu'il s'agit, comme celle que Leibniz évoque dans ce passage : "Et celles-ci [ces vérités nécessaires] sont des vérités éternelles, car non seulement elles tiendront aussi longtemps que le monde durera, mais encore elles tiendraient si Dieu avait créé le monde selon quelque autre plan" [Leibniz 03], p. 18.

Enfin, en 1963, Kripke introduit un troisième concept de monde possible :

(1) Les individus peuvent varier d'un monde à l'autre.

(2) La nécessité n'est plus une notion absolue mais bien une notion relative.

Un état de chose nécessaire dans un monde H n'est plus un état de chose réalisé dans tous les mondes possibles H', mais dans tous les mondes H' *accessibles* à partir de H.

[1] Une *description d'état* est un ensemble d'énoncés qui contient, pour tout énoncé atomique du langage construit à l'aide d'un de ces symboles de prédicat, soit cet énoncé, soit sa négation (mais pas les deux), et ne contient rien d'autre.

9.1.1. *La formule de Barcan et sa converse*

Nous avons eu l'occasion d'examiner antérieurement la deuxième innovation. Examinons à présent la première.

Pour mesurer l'enjeu de l'adoption d'une sémantique où tous les mondes possibles d'un modèle cessent de contenir les mêmes individus, il est éclairant d'examiner une formule qui est valide dans la sémantique où tous les mondes possibles d'un modèle contiennent les *mêmes individus* et qui cesse de l'être quand ce n'est plus le cas.

La formule :

$$\forall x \, \Box Px \equiv \Box \forall x \, Px,$$

permet de voir concrètement la portée du changement intervenu lors du passage du deuxième au troisième concept de monde possible.

Cette formule résulte de la conjonction de la formule de Barcan (B.F.) :

$$\forall x \, \Box Px \supset \Box \forall x \, Px,$$

et de sa converse (F.B.) ou (C.F.B) :

$$\Box \forall x \, Px \supset \forall x \, \Box Px.$$

Les médiévaux connaissaient la différence entre la modalité *de dicto* (qui porte sur la proposition) illustrée par "nécessairement tout x est P" et la modalité *de re* (qui porte sur la chose), illustrée par "tout x est nécessairement P" [Prior 62], p. 209-210.

Prenons l'instance suivante de (F.B.) : *Si nécessairement chaque objet blanc est blanc, alors chaque objet blanc est nécessairement blanc.*

Cette proposition est suspecte : *l'antécédent* semble aller de soi, mais *le conséquent* est discutable. Une sémantique reposant sur une notion de monde possible qui nous oblige à admettre la converse de la loi de Barcan est dès lors également suspecte. Au contraire, la sémantique reposant sur le troisième concept de monde possible nous libère de cette obligation.

9.1.2. *Sémantique de la logique modale quantifiée avec domaine variable*

Pour formuler une sémantique satisfaisante de la logique modale quantifiée, il faut faire appel à un modèle plus complexe que les modèles utilisés en logique propositionnelle, à savoir à un quintuplet (cfr. § 3.2.1) :

$$\mathcal{M} = <W, R, S, Q, V>,$$

où Q est le domaine de quantification et S le domaine du discours. Dans l'article fondateur de Kripke [Kripke 63], le domaine S varie avec les mondes du modèle. On peut voir en $S(w)$ une fonction qui à tout monde possible w

associe les individus qui existent en w, en d'autres termes, une fonction qui associe à w le domaine de w.

Le domaine Q – que nous avons appelé *domaine quantificationnel* – est la réunion de tous les domaines de la *structure* considérée. On l'appelle pour cette raison *domaine de la structure.* Pour interpréter un n-uplet de variables, on aura besoin d'un n-uplet de valeurs prélevées dans le produit cartésien $Q_1 \times \ldots \times Q_n$ que l'on dénote par Q.

Il ne suffit pas d'avoir une *structure* $\mathcal{F}$ pour interpréter une formule de la logique modale quantifiée. Il faut aussi une *valuation* V qui, jointe à $\mathcal{F}$, nous donne un *modèle* $\mathcal{M}$. Pour pouvoir évaluer une formule de cette logique, on a besoin d'ajouter aux clauses de la définition récursive de la vérité, énoncées au paragraphe 2.2.3, deux clauses de plus, l'une pour interpréter les formules contenant des variables libres, l'autre pour interpréter les formules contenant des variables quantifiées universellement.

Soit g une assignation *d'éléments* $b_1 \ldots b_n$ de Q à $y_1 \ldots y_n$, $\mathcal{I}$ une interprétation du symbole de relation et g' une assignation *d'éléments* de $S(w)$ à x.

$$\mathcal{M} \models_w \forall x\, (Px, y_1 \ldots y_n) \Leftrightarrow \mathcal{M} \models_w Px, y_1 \ldots y_n,$$

pour toute assignation d'individus $b_1 \ldots b_n$, pris dans le domaine quantificationnel Q, à $y_1 \ldots y_n$, et pour toute assignation d'un individu a, pris dans le domaine du monde w, c'est-à-dire $S(w)$, à x.

Soit g' une assignation d'éléments $a_1 \ldots a_n$ de Q à $x_1 \ldots x_n$ et $\mathcal{I}$ une interprétation du symbole de relation P par la relation $\mathcal{P}$;

$$\mathcal{M} \models_w P^n(x_1 \ldots x_n) \Leftrightarrow \mathcal{M} \models_w P^n(a_1 \ldots a_n).$$

Remarque 1. L'emploi de a_i dans le membre de droite est un abus de langage car ce ne sont pas des noms du langage-objet.

Remarque 2. On puise les valeurs des variables liées par le quantificateur dans le domaine du monde dans lequel est évaluée la formule quantifiée, mais on puise les valeurs des variables libres dans le domaine de la structure.

Comme le note Kripke, "nous quantifions seulement sur les objets qui existent réellement en w" [Kripke 63], p. 67. Au contraire, $\mathcal{M} \models_w Px \vee \neg Px$ même si $A(x)$ n'existe pas en w. Les exigences pour les variables libres sont donc moins fortes. Il suffit que $g(x)$ existe dans l'un des mondes de la structure $\mathcal{F}$.

9.1.3. *Modèles falsificateurs pour la formule de Barcan*

Considérons les versions duales de la formule de Barcan et de sa converse à savoir :

(1) $\Diamond\exists x\, Px \supset \exists x \Diamond Px$ (Formule de Barcan).

(2) $\exists x \Diamond Px \supset \Diamond\exists x\, Px$ (Converse de la formule de Barcan).

On peut illustrer ces formules par les instances suivantes, $\Diamond$ étant pris dans le sens temporel :

(3) *S'[il est vrai qu'] il y aura un roi, il y a un futur roi.*

(4) *S'il y a un futur roi, il y aura un roi.*

Si nous interrogeons notre intuition, nous dirons probablement que (3) est fausse et que (4) est vraie. La proposition *"Il y aura un roi"* paraît avoir des conditions de vérité moins contraignantes que *"Il y a un futur roi"*. La deuxième proposition n'est vraie que s'il existe déjà dans le présent une valeur de x qui satisfera "x est un roi" dans le futur.

Fitch reconnaît le statut particulier de la formule de Barcan, mais se prononce sur sa validité avec prudence : "Il n'est pas possible de dériver $\Box\forall x Fx$ de $\forall x \Box Fx$ à moins d'adopter une règle spéciale à cet effet. Une telle règle ne violerait pas la consistance du système et il semblerait qu'elle soit un principe valide" [Fitch 52], p. 165.

Le recours à la sémantique formelle permet de mettre un terme à nos hésitations et de fournir une réponse ferme, précise et bien fondée, tant pour la formule de Barcan que pour sa converse. La formule de Barcan est valide dans les modèles à *domaine constant* et cesse d'être valide dans les modèles à *domaines variables* dans lesquels, lors de la transition d'un monde à un autre, le *domaine* d'individus *s'accroît*. La converse de la formule de Barcan est valide dans les modèles à *domaine constant* et cesse d'être valide dans les *modèles* à *domaines variables* lorsque, lors de la transition d'un monde à un autre, le domaine d'individus décroît.

Formellement, on représente ainsi ces exigences [Kripke 63], p. 67-68 :

(1) La formule de Barcan, dans un domaine variable, est valide seulement si la condition suivante est satisfaite :

Si $w_1 R w_2$, alors $S(w_2) \subseteq S(w_1)$. Condition d'*antimonotonie*.

(2) La converse de Barcan, dans un domaine variable, est valide seulement si la condition suivante est satisfaite :

Si $w_1 R w_2$, alors $S(w_1) \subseteq S(w_2)$. Condition de *monotonie*.

9.1.4. *Modèle falsificateur pour la converse de la formule de Barcan*

Soit $<W, R, S, Q, V>$ où :
$W = \{w_1\ w_2\}$;
$R = \{<w_1, w_2>\}$;
$S(w_1) = \{a, b\}, S(w_2) = \{a\}$;
$Q = \{a, b\} \cup \{a\} = \{a, b\}$;
$V(Pb, w_2) = \mathbf{V}$.

Ce modèle ne satisfait pas la clause de monotonie. En effet, on a $w_1 R w_2$ et $S(w_1) \not\subseteq S(w_2)$.

Démontrons que $\mathcal{M} \not\models_{w_1} \exists x \Diamond Px \supset \Diamond \exists x\, Px$ (exercice proposé par Fitting [Fitting and Mendelsohn 98], p. 110). Si on interprète $\Diamond$ par l'opérateur du futur et si on donne à P le sens de "absent", la tâche à accomplir est très claire : il s'agit de construire un modèle qui vérifie "Il y a quelqu'un qui sera absent" et qui falsifie "Il sera le cas que quelqu'un est absent".

(A) Supposons vrai l'antécédent ($\exists x \Diamond Px$). En vertu des clauses de la définition sémantique de la satisfaction (et de la vérité) pour la logique modale quantifiée avec domaines variables, on a [g et g' sont des assignations] :

- $\mathcal{M} \models_{w_1, g} \exists x \Diamond Px$ ssi $\mathcal{M} \models_{w_1, g'} \Diamond Px$ pour au moins une assignation g' dans $S(w_1)$ qui est une x-variante de g. Soit $g(x) = a$ et $g'(x) = b$).
- $\mathcal{M} \models_{w_1, g'} \Diamond Px$ ssi $\mathcal{M} \models_{w_2, g'} Px$. Cette condition est satisfaite puisque $w_1 R w_2$ et que $V(Pb, w_2) = \mathbf{V}$.

(B) Supposons vrai le conséquent ($\Diamond \exists x\, Px$) :

- $\mathcal{M} \models_{w_1, g} \Diamond \exists x\, Px$ ssi $\mathcal{M} \models_{w_2, g} \exists x\, Px$;
- $\mathcal{M} \models_{w_2, g} \exists x\, Px$ ssi $w_1 R w_2$ et $\mathcal{M} \models_{w_2, g'} Px$ pour au moins une assignation g' dans $S(w_2)$ qui est une x-variante de a. La condition n'est pas satisfaite. La seule x-variante de a, c'est a puisque le domaine de w_1 ne contient que a. Or, selon la valuation V, Px n'est satisfaite dans w_2 que par b.

(C) On a donc établi que, dans le modèle $\mathcal{M}$ considéré, l'antécédent de la formule F.B. est vrai et le conséquent faux. La formule B.F. est donc rendue fausse par le modèle $\mathcal{M}$.

Le lecteur est invité à construire un modèle à domaines variables qui rende fausse la formule de Barcan.

9.1.5. *Les mérites respectifs des modèles avec domaine constant et des modèles avec domaines variables*

Dans les modèles à domaines variables dont on vient d'exposer la sémantique, on ne quantifie que sur les individus qui *existent* dans le monde par rapport

auquel on évalue la formule modale. Le quantificateur existentiel, dans ces conditions, exprime bien l'existence au sens courant.

Dans la sémantique à domaine constant (ou fixe), les quantificateurs portent non sur ce qui *existe*, mais sur ce qui *pourrait exister*. La lecture du quantificateur existentiel n'est plus "Il existe un tel et tel" mais "Il y a un x, dont il se pourrait qu'il existe tel que ... " [Fitting and Mendelsohn 98], p. 106.

Si le quantificateur existentiel n'exprime plus *l'existence réelle* mais seulement *l'existence possible*, nous devons – pour ne pas appauvrir l'expressivité de la langue formelle – inventer un nouveau symbole pour exprimer l'existence. Par exemple, nous pouvons introduire un prédicat d'existence.

9.1.6. *Immersion de la sémantique à domaines variables dans la sémantique à domaine constant*

Hughes et Cresswell montrent comment concilier la *simplicité* de la sémantique à domaine constant et l'*expressivité* de la sémantique à domaines variables [Hughes and Cresswell 96], p. 303. Leur solution consiste à dédoubler les quantificateurs par l'adoption conjointe des quantificateurs appelés "possibilistes" dont le domaine est l'ensemble des individus du modèle (c'est-à-dire dans la réunion des domaines du modèle) et des quantificateurs appelés "réalistes" (*actualist* en anglais) dont le domaine est l'ensemble des individus existant dans un domaine distingué : le domaine du monde réel; nous avons :

(A) $\forall x\, Px$ (quantificateur possibiliste);

(B) $\Pi x\, Px$ (quantificateur réaliste).

Le quantificateur réaliste universel peut être défini à partir du quantificateur possibiliste et d'un prédicat d'existence adopté comme *terme primitif* :

(C) $\Pi x\, Px =_{def} \forall x\, (Ex \supset Px)$;

Il en va de même pour le quantificateur réaliste existentiel Σx :

(D) $\Sigma x\, Px =_{def} \exists x\, (Ex \wedge Px)$.

Hughes et Cresswell (1996) ont réussi à immerger [*embed*] la sémantique à domaines variables dans la sémantique à domaine constant de manière telle que la seconde puisse simuler la première grâce aux quantificateurs restreints au moyen du prédicat d'existence.

9.1.7. *Immersion de la sémantique à domaine constant dans la sémantique à domaines variables*

Fitting et Mendelsohn (1998) ont réussi l'opération d'immersion inverse. Contrairement à Hughes et Cresswell ils prennent le quantificateur réaliste

comme primitif. Selon cette sémantique, pour chaque monde w, le domaine $S(w)$ de w est constitué des individus qui existent dans w et le champ des variables liées par les quantificateurs est le domaine de w. Ultérieurement Fitting et Mendelsohn ont développé les outils pour pouvoir parler sur "le mode probabiliste". [Fitting and Mendelsohn 98], p. 163, c'est-à-dire pour pouvoir *parler de choses qui n'existent pas*, tout en utilisant comme quantificateurs de base les quantificateurs réalistes.

Leur stratégie consiste à utiliser une variété de logique libre dans laquelle les variables libres ne sont pas tenues de désigner (d'avoir des valeurs). Pour qu'elles désignent, il faut affirmer explicitement qu'elles le font. Cette affirmation se fait au moyen d'un postulat utilisant le prédicat d'existence Ex.

Contrairement à Hughes et Cresswell, Fitting et Mendelsohn définissent le prédicat d'existence. Ils ne le prennent pas comme primitif et, par surcroît, le définissent à l'aide du quantificateur réaliste $\exists x$:

$$Ex =_{def} \exists y\,(y = x).$$

9.1.8. *Evaluation des deux types d'unification des sémantiques*

Les deux sémantiques – celle de Hughes et Cresswell et celle de Fitting – ont la même souplesse et la même puissance. L'une et l'autre nous donnent la possibilité de parler de ce qui n'existe pas sans tomber dans l'incohérence, et la possibilité d'accepter ou de rejeter la formule de Barcan et sa converse.

Le choix des quantificateurs réalistes comme quantificateurs de base préconisé par Fitting est incontestablement plus proche de l'usage courant. Si quelqu'un nous dit *"Il existe des fées"*, nous le prenons au pied de la lettre, à moins qu'il n'ajoute *"dans les contes de fées"*. Cet ajout, cependant, ne prouve nullement qu'il y a deux *sortes d'existence* et *deux sens* du mot "exister" : "exister dans le monde" et "exister dans l'imagination". Il prouve seulement qu'il est possible de parler de quelque chose sans assumer son existence. Nous renvoyons le lecteur à l'article de Maria Reyes [Reyes 94] et au traitement du prédicat "Désigne" chez Fitting et Mendelsohn.

9.1.9. *Expression syntaxique de la monotonie et de l'antimonotonie*

Le prédicat d'existence "E" permet d'*exprimer syntaxiquement* les *propriétés sémantiques* de monotonie et d'antimonotonie :

La structure $\mathcal{F}$ est monotone si et seulement si $Ex \supset \Box Ex$.

La structure $\mathcal{F}$ est antimonotone si et seulement si $\Diamond Ex \supset Ex$.

Théorème 9.1. Si la relation d'accessibilité est symétrique, la monotonie et l'antimonotonie s'impliquent mutuellement [Fitting and Mendelsohn 98], p. 185.

Démonstration. Démontrons que la monotonie exprimée par $Ex \supset \Box Ex$ et la symétrie exprimée par l'axiome B impliquent l'antimonotonie.

1. $Ex \supset \Box Ex$: monotonie.
2. $\Diamond Ex \supset \Diamond\Box Ex$: règle de K : $A \supset B \vdash \Diamond A \supset \Diamond B$.
3. $\neg p \supset \Box\Diamond\neg p$: axiome B, $\neg p/p$.
4. $\neg\Box\Diamond\neg p \supset \neg\neg p$: contraposition sur 3.
5. $\Diamond\Box p \supset \neg\neg p$: définition de $\Box$ et de $\Diamond$.
6. $\Diamond\Box p \supset p$: double négation.
7. $\Diamond\Box Ex \supset Ex$: 6, Ex/p.
8. $\Diamond Ex \supset Ex$: de 2 et 7 par syllogisme.

Exercice. Démontrez la réciproque (solution dans [Fitting and Mendelsohn 98]).

Remarque. L'existence d'un lien entre des propriétés du domaine (monotonie et antimonotonie) et une propriété de la relation d'accessibilité (symétrie) ne va pas de soi.

9.2. La preuve par tableaux en logique modale quantifiée

La méthode des tableaux utilisée par M. Fitting est une méthode des tableaux explicites comme celle, postérieure dans le temps, de Nerode et Shore. Il y a de légères variantes notationnelles. Les mondes figurant dans les préfixes des formules chez Fitting sont représentés par des suites de chiffres : si σ est un préfixe et n est un entier positif, $\sigma.n$ est σ suivi d'un point suivi de n [Fitting and Mendelsohn 98], p. 48.

Par exemple, dans le schéma de tableau ci-dessous, 1, 1.1, 1.2 et 1.1.1 sont des préfixes :

(1) $1.X$;

(2) $1.1Y$;

(3) $1.2Z$;

(4) $1.1.1U$.

Les préfixes 1.1 et 1.2 désignent deux mondes accessibles à partir du monde 1. Le préfixe 1.1.1 désigne un monde accessible à partir du monde 1.1, lui-même accessible à partir du monde 1.

9.2.1. *Règles de réduction pour les opérateurs modaux*

Définition 9.1. Nous appellerons *ligne-prémisse* la ligne figurant dans le

numérateur et *ligne-conclusion* la ligne figurant dans le dénominateur.

Règles pour les formules π communes à tous les systèmes :

$$\frac{\sigma \Diamond X}{\sigma.n\ X};\qquad \frac{\sigma \neg \Box X}{\sigma.n\ \neg X}.$$

Condition requise : $\sigma.n$ est nouveau sur la branche.

Règles pour les formules ν communes à tous les systèmes :

$$\frac{\sigma \Box X}{\sigma.n\ X};\qquad \frac{\sigma \neg \Diamond X}{\sigma.n\ \neg X}.$$

Condition requise : $\sigma.n$ figure déjà sur la branche.

Règles pour les formules ν particulières à certains systèmes :

Pour les préfixes σ et $\sigma.n$ déjà présents, les trois règles suivantes concernent respectivement les axiomes T, B et 4.

$$\frac{\sigma \Box X}{\sigma X};\qquad \frac{\sigma \neg \Diamond X}{\sigma \neg X}.$$

$$\frac{\sigma.n \Box X}{\sigma X};\qquad \frac{\sigma n \neg \Diamond X}{\sigma \neg X}.$$

$$\frac{\sigma \Box X}{\sigma.n \Box X};\qquad \frac{\sigma \neg \Diamond X}{\sigma.n \neg \Diamond X}.$$

Exemple de preuve en $K4$:

(1)	1	$\neg(\Diamond\Diamond p \supset \Diamond p)$		
(2)	1	$\Diamond\Diamond p$	(1)	élimination de $\supset$.
(3)	1	$\neg\Diamond p$	(1)	élimination de $\supset$.
(4)	1.1	$\neg\Diamond p$	(3)	Règle ν propre au système 4.
(5)	1.1	$\Diamond p$	(2)	élimination de $\Diamond$.
(6)	1.1.1	p	(5)	élimination de $\Diamond$.
(7)	1.1.1	$\neg p$	(4)	élimination de $\neg\Diamond$.

9.2.2. *Règles de réduction pour les quantificateurs*

Règles pour les propositions universelles dans une logique correspondant à une sémantique à domaines variables :

La procédure de preuve par tableaux décrite par Fitting [Fitting and Mendelsohn 98], p. 116-131 est conçue pour démontrer des formules fermées. On se donne des variables d'instantiation à utiliser dans le corps de la preuve appelées paramètres. Ces paramètres sont *indexés* sur le monde désigné par le préfixe de la formule où ils apparaissent. Le paramètre a_σ, par exemple, est associé au préfixe σ de la formule $\sigma.X$.

Les règles γ pour les universelles s'énoncent ainsi :

$$\frac{\sigma \forall x\ Px}{\sigma P a_\sigma};\qquad \frac{\sigma \neg \exists x\ Px}{\sigma \neg P a_\sigma}.$$

Dans ces expressions, a_σ est n'importe quel paramètre de la liste de paramètres donnée au départ, avec les règles de formation, respectant la contrainte suivante : l'indice du paramètre est le même que celui du préfixe de la formule où il apparaît [Fitting 83], p. 433 et [Fitting and Mendelsohn 98], p. 120, Une technique apparentée est utilisée par Ramsay [Ramsay 88], p. 142. Voir aussi P. Jackson et H. Reichgelt [Jackson et al., 89a].

Les règles δ pour les formules existentielles s'énoncent ainsi :

$$\frac{\sigma \exists x\, Px}{\sigma Pa_\sigma}; \qquad \frac{\sigma \neg \forall x\, Px}{\sigma \neg Pa_\sigma}.$$

où le paramètre a_σ est nouveau sur la branche.

9.2.3. *Exemple de démonstration en QK*

$$\vdash \Diamond \exists x\, (Px \supset \Box q) \supset \Diamond(\forall x\, Px \supset \Box q).$$

(1)	1	$\neg[\Diamond \exists x\, (Px \supset \Box q) \supset \Diamond(\forall x\, Px \supset \Box q)]$	
(2)	1	$\Diamond \exists x\, (Px \supset \Box q)$	de 1 par $\supset$-él.;
(3)	1	$\neg \Diamond(\forall x\, Px \supset \Box q)$	de 1 par $\supset$-él.;
(4)	1 1	$\exists x\, (Px \supset \Box q)$	de 2 par π;
(5)	1 1	$\neg(\forall x\, Px \supset \Box q)$	de 3 par ν;
(6)	1 1	$Pa_{11} \supset \Box q$	de 4 par δ;
(7)	1 1	$\forall x\, Px$	de 5 par $\supset$-él.;
(8)	1 1	$\neg \Box q$	de 5 par $\supset$-él.;
(9)	1 1	Pa_{11}	de 7 par γ;
(10)	1 1	$\neg Pa_{11}$ | $\Box q$	de 6 par $\supset$-él.;
(11)	1 1 1	$\bot$ | $\neg q$	de 8 par π;
(12)	1 1 1	q | $\bot$	de 10 par ν.

9.2.4. *Exemple de non-théorème de QK*

$$\nvdash \Box \exists x\, Px \supset \exists x\, \Box Px.$$

(1)	1	$\neg(\Box \exists x\, Px \supset \exists x\, \Box Px)$	
(2)	1	$\Box \exists x\, Px$	de (1) par $\supset$-él.;
(3)	1	$\neg \exists x\, \Box Px$	de (1) par $\supset$-él.;
(4)	1	$\neg \Box Pa_1$	de (3) par γ;
(5)	1.1	$\neg Pa_1$	de (4) par π;
(6)	1.1	$\exists x\, Px$	de (5) par ν;
(7)	1.1	$Pb_{1.1}$	de (6) par δ.

Le tableau ne se ferme pas. Il est impossible d'unifier (5) et (7) à cause des paramètres différents (a_1 et $b_{1\,1}$).

9.3. Correction et complétude

9.3.1. *Généralités*

On se bornera à des indications succinctes.

Correction. On définit la notion de consistance sémantique (*satisfiability*) dans un modèle à domaines variables relativement à une valuation. On montre ensuite que les extensions de branches obtenues par application des règles $\alpha, \beta, \gamma, \delta, \pi, \nu$ préservent la consistance sémantique.

Complétude. On veut prouver que pour toute formule fermée sémantiquement inconsistante, il *existe* un tableau fini fermé. On prouve la contraposée, à savoir que si on a échoué à obtenir un tableau fermé (dont toutes les branches aboutissent à $\perp$), on peut extraire d'une branche ouverte un modèle falsificateur.

Mais n'importe quelle branche ouverte ne fait pas l'affaire : "nous pourrions ... être passés à coté de la preuve" [Fitting and Mendelsohn 98], p. 125. Pour écarter cette possibilité, on définit une procédure de *construction systématique* [Fitting and Mendelsohn 98], p. 126-127, [Nerode and Shore 93], p. 228-230. Cette procédure qui ne fait appel à aucun moment à l'inventivité ou à la créativité possède la propriété d'*équité* : toutes les manières possibles d'étendre les branches ont été essayées tôt ou tard.

Comme par hypothèse la forme X est supposée non prouvable, il doit exister une branche ouverte, finie ou infinie. Cette branche ouverte, comme toutes les branches de tableau, est un ensemble saturé vers le bas.

9.3.2. *Extraction d'un modèle falsificateur*

Pour extraire un modèle falsificateur de la formule initiale "$1\neg X$", on se donne un ensemble W où les mondes sont les préfixes d'une branche ouverte $\sigma_1, \ldots, \sigma_n$ et on se donne comme domaine $\mathcal{D}$ l'ensemble des paramètres. La fonction d'assignation g envoie chaque paramètre sur lui-même :

$$g(a_i) = a. \tag{9.1}$$

Enfin, on interprète tout symbole de relation "$R(t_1, \ldots, t_n)$" comme dénotant la relation entre les objets assignés par g aux paramètres "t_1", ... ,"t_n".

On prouve alors par induction le lemme fondamental suivant :

Lemme 9.2. Si une formule préfixée est positive sur la branche $\mathcal{B}$, alors σX est vraie dans le modèle $\mathcal{M}$, au monde σ, pour l'assignation g. Formellement :

$$\sigma X \Rightarrow \mathcal{M} \models_{\sigma,g} X. \tag{9.2}$$

Lemme 9.3. Si une formule préfixée est négative sur la branche $\mathcal{B}$, alors $\sigma \neg X$ est fausse dans ce même modèle au monde σ, pour l'assignation g.

Formellement :

(9.3) $\sigma \neg X \Rightarrow \mathcal{M} \not\models_{\sigma,g} X.$

Pour la preuve par induction, nous renvoyons le lecteur à [Fitting and Mendelsohn 98], p. 127-128.

Le point sur lequel il faut insister c'est que, bien que généralement infini, ce modèle est *constructif*. Le contre-exemple falsificateur de la formule "$1\neg X$" située à la racine est extrait d'un tableau dans lequel "chaque étape détermine exactement celle qui suit".

L'intérêt majeur de la preuve de complétude que l'on vient de décrire réside précisément dans son caractère *constructif* et dans la possibilité qu'elle nous donne d'extraire méthodiquement un modèle falsificateur d'un branche ouverte. Contrairement à ce qui se passe pour la logique modale propositionnelle, on ne peut tirer de la *preuve de complétude constructive* que l'on vient de présenter, une *procédure de décision* : "A un stade fini [de la construction systématique du modèle] nous ne savons généralement pas distinguer les branches qui se fermeront des branches qui continueront à croître ... , ce qui signifie que nous ne devons pas nous attendre à tirer de notre procédure systématique de construction de tableaux, une manière de *décider* de la prouvabilité ou de la non-prouvabilité des formules" [Fitting and Mendelsohn 98], p. 128.

Nous ne devons pas nous en étonner puisque la logique des prédicats n'est que semi-décidable (théorème de Church 1936).

9.4. Logique modale quantifiée avec identité

En logique non modale, on obtient la logique du premier ordre avec identité en ajoutant aux axiomes et aux règles gouvernant les connecteurs et les quantificateurs les deux principes suivants gouvernant le prédicat binaire d'identité :

(9.4) $x = x$: réflexivité de l'identité

(9.5) $x = y \supset Px \equiv Py$: indiscernabilité des identiques

La première formule est un axiome. La deuxième est un schéma d'axiome. Il faut distinguer la loi de Leibniz (9.5) du principe affirmant l'identité des indiscernables (9.6) (également dû à Leibniz). Ce principe est la réciproque du schéma (9.5), mais, contrairement à celui-ci, il appartient à la logique du deuxième ordre. Il énonce que deux individus qui ont toutes leurs propriétés en commun sont identiques :

(9.6) $\forall P\,(Px \equiv Py \supset x = y).$

Considérons la variante (9.7) de (9.5) :

(9.7) $x = y \supset (Px \supset Py).$

Substituons alors à P le prédicat $\Box(x = ①)$, nous obtenons en remplaçant ① par le premier terme qui suit P dans Px (respectivement Py) :

(9.8) $\quad [x = y \supset (\Box(x = x) \supset \Box(x = y))]$,

et par permutation des antécédents :

(9.9) $\quad [\Box(x = x) \supset (x = y \supset \Box(x = y))]$.

Comme $\Box(x = x)$ s'obtient par *nécessitation* à partir de (9.4), on obtient par *modus ponens* :

(9.10) $\quad x = y \supset \Box(x = y)$.

La formule (9.10) exprime la proposition que si deux êtres sont identiques, ils sont nécessairement identiques.

De prime abord, cette proposition nous heurte. Nous désirons distinguer les identités nécessaires des identités contingentes.

On peut cependant reconnaître le bien-fondé de la distinction entre *identité contingente*, telle que l'identité de Brutus et du meurtrier de César, et *identité nécessaire*, telle que l'identité de 4 et du carré de 2, tout en affirmant qu'elle n'est pas menacée par (9.10). On peut soutenir, comme le font plusieurs auteurs, que cette distinction ne peut être exprimée dans un langage appauvri dont les seuls termes individuels sont des variables ou des paramètres. Elle n'apparaît que dans des formules qui contiennent des noms propres et des descriptions définies.

Une autre objection a été soulevée à propos de (9.10). Exprimée en langue naturelle, en faisant usage de la terminologie des mondes possibles, la formule (9.10) serait rendue par la phrase (9.11) :

(9.11) Si $x = y$ alors dans tous les mondes possibles $x = y$.

Nous avons l'habitude de reconnaître les individus à travers les moments du temps ou de suivre leurs déplacements dans l'espace. Il en va tout autrement de l'identification à travers les mondes possibles. Comme le note Quine : "La différence dévastatrice [entre les deux premiers cas de ré-identification et le dernier] est que la série des coupes transversales de notre monde réel est imposée à nous ... tandis que tous les chemins de gradation continue d'un monde possible à l'autre sont le fruit d'un choix arbitraire" [Quine 76, 81], p. 127.

Nous mentionnons ce problème philosophique important sans discuter les solutions qui lui ont été apportées et qui sortent du cadre de cet ouvrage. D'un point de vue technique, il ne met pas en cause la notion d'identité en logique modale quantifiée si on se limite à la logique temporelle quantifiée dont l'interprétation requiert des moments du temps plutôt que des mondes possibles. Le contraste entre :

"Il y aura un roi", ce qui se représente par :

(9.12) $F\,\exists x\,Gx$,

et : "Il y a un futur roi", qui s'exprime par :

(9.13) $\exists x\,F\,Gx$,

est très clair. Dans (9.13), le futur roi est déjà né.

Pour obtenir une procédure de preuve par tableaux pour la logique du premier ordre avec égalité, il faut ajouter deux règles qui correspondent aux axiomes 9.4 et 9.5 [Fitting and Mendelsohn 98], p. 150-151.

9.4.1. *Règles de réduction pour la logique de l'identité*

(1) **La règle de réflexivité :**

Si a est un paramètre et σ est un préfixe qui apparaît plus haut sur la branche,

$$\frac{}{\sigma(a = a)},$$

peut être ajouté à la fin de la branche.

(2) **La règle de substitutivité des identiques :**

$$\frac{\sigma_1(a = b) \quad \sigma_1 Pa}{\sigma_2 Pb}$$

$$\frac{\sigma_1(b = a) \quad \sigma_1 Pa}{\sigma_2 Pb}$$

9.5. La preuve de complétude de K quantifié avec identité

La preuve de complétude de la logique modale quantifiée *avec identité* est un développement parfaitement homogène de la preuve de complétude de la logique modale quantifiée *sans identité*. Fitting et Mendelsohn étendent la procédure de construction systématique de tableaux de manière à prendre en compte les nouvelles règles gouvernant le prédicat d'identité. Ils montrent ensuite que si la procédure systématique échoue à engendrer un tableau fini fermé, elle engendre un tableau ouvert, généralement infini, dont on peut extraire un contre-exemple à la formule niée placée à la racine.

Comme dans le cas de la logique modale quantifiée sans identité, on se donne un domaine "linguistique". La fonction d'assignation g envoie chaque paramètre sur lui-même et on se donne un ensemble de mondes "linguistiques". Le fonction Θ envoie chaque préfixe σ sur lui-même. On montre ensuite les

deux implications suivantes, comme dans le cas de la preuve de complétude du système K avec quantificateurs, mais sans symbole d'identité :

(9.14) σX sur $\mathcal{B} \Rightarrow \mathcal{M} \models_{\sigma,g} X$;

(9.15) $\sigma \neg X$ sur $\mathcal{B} \Rightarrow \mathcal{M} \not\models_{\sigma,g} X$.

Une difficulté propre à la logique de l'identité survient lorsque la branche contient des formules positives de la forme :

(9.16) $\sigma\, a_{11} = b_1$.

En effet, la clause (9.14) ci-dessus, nous conduit à dire que dans le modèle dont le domaine est constitué par les paramètres a_{11}, b_1, etc, ... , la formule "$a_{11} = b_1$", qui exprime que a_{11} est identique à b_1, évaluée au monde désigné par le préfixe "σ" est vraie, bien que a_{11} et b_1 soient des *objets différents.*

On doit en conclure que le signe "=" n'est pas pris dans le sens *normal,* mais dans le sens, plus général, de *relation d'équivalence.* Des objets distincts peuvent en effet être reliés par une *relation d'équivalence* comme : "x a la même taille que y" *sans être identiques.*

Les auteurs précités ont résolu le problème en montrant qu'il est possible de convertir le modèle non normal en un modèle normal. Il suffit de diviser le domaine constitué de paramètres par la relation d'équivalence initialement exprimée par le symbole "=" et d'assigner les classes d'équivalence comme objets dénotés (*dénotation*) aux paramètres. Cette technique a déjà été utilisée dans la preuve de la complétude du calcul des prédicats avec identité de Henkin en 1949.

9.6. Logique modale quantifiée avec prédicat d'existence

On a introduit plus haut le prédicat d'existence $E(x)$ par une définition rappelée ci-dessous :

(9.17) $E(x) =_{def} \exists y\,(x = y)$.

Cette définition permet de dériver deux règles d'inférence pour la logique modale quantifiée [Fitting and Mendelsohn 98], p. 165-166.

La première règle est une extension de la règle d'instantiation universelle. Jusqu'à présent, la règle d'instantiation exigeait que le souscrit du paramètre dans la formule instanciée fût le même que le préfixe de la formule :

$$\frac{\sigma \forall x\, Px}{\sigma\, Pa_\sigma}$$

A présent, cette contrainte pourra être assouplie si l'on dispose de la prémisse d'existence appropriée :

$$\frac{\sigma\,\forall x\,Px \qquad \sigma\,E(a_\tau)}{\sigma\,Pa_\tau}$$

où $E(a_\tau)$ est une abréviation de $\sigma(\exists y)(y = a_\tau)$.

La deuxième règle autorise la fermeture d'une branche dès qu'apparaît une formule de la forme :

(9.18) $\sigma \neg E(a_\sigma)$.

Justification.

$$\begin{aligned} &\sigma \neg E(a_\sigma);\\ &\sigma \neg \exists y(y = a_\sigma);\\ &\sigma \forall y \neg(y = a_\sigma);\\ &\sigma \neg(a_\sigma = a_\sigma). \end{aligned}$$

La dernière ligne contredit le premier axiome de l'identité.

Exercice. Justifiez la première règle.

Exemples.

$\vdash x = y \supset y = x$ (Symétrie de l'identité).

On utilise la règle de réflexivité et la règle de substitutivité des identiques. Dans cette dernière, on emploie la substitution $\textcircled{1} = x\,/P$.

$$\left[\begin{array}{c} \sigma : x = y \\ \sigma : Px \\ \hline \sigma : Py \end{array}\right] \quad \text{devient} \quad \left[\begin{array}{c} \sigma : x = y \\ \sigma : x = x \\ \hline \sigma : y = x \end{array}\right] \qquad \begin{array}{l}(a)\\(b)\\(c)\end{array}$$

(1)	1	$\neg(x = y \supset y = x)$;	
(2)	1	$x = y$	de (1) par $\supset$-él.;
(3)	1	$\neg(y = x)$	de (1) par $\supset$-él.;
(4)	1	$x = x$	par la règle de réflexivité;
(5)	1	$y = x$	de (2) et (4) par la règle de substitutivité des identiques dans laquelle Px devient $x = x$ et Py devient $y = x$.

$\perp$

$\vdash \forall x \forall y\,[\Diamond(x = y) \supset (y = x)]$.

(1)	1	$\neg\forall x\,\forall y\,[\Diamond(x = y) \supset (y = x)]$;	
(2)	1	$\neg\forall y\,[\Diamond(a = y) \supset (y = a)]$	de (1) par δ;
(3)	1	$\neg[\Diamond(a = b) \supset (b = a)]$	de (2) par δ;
(4)	1	$\Diamond(a = b)$	de (3) par $\supset$-él.;
(5)	1	$\neg(b = a)$	de (3) par $\supset$-él.;
(6)	1.1	$a = b$	de (4) par π;
(7)	1	$\Box\neg(b = a)$	par $\neg(x = y) \supset \Box\neg(x = y)$;
(8)	1.1	$\neg(b = a)$	de (7) par ν;
(9)	1.1	$\neg(a = b)$	de (8) par symétrie de l'identité.
		$\perp$	

Le principe utilisé en (7) est admis par Fitting et Mendelsohn : " ... si x et y sont les mêmes, le fait est nécessaire ... Qu'en est-il si x et y sont distincts ?" Cette non-identité est-elle nécessaire aussi? Nous tenons que la réponse est "oui". Deux objets ne peuvent, même si les circonstances changent, devenir un [Fitting and Mendelsohn 98], p. 146.

Hughes et Cresswell ne jugent pas le principe de la non-identité nécessaire moins crédible que le principe de l'identité nécessaire : "Quoi que l'on pense de l'identité nécessaire, il est à tout le moins défendable [d'affirmer] que si un système modal satisfaisant contient l'identité nécessaire, il doit aussi contenir la différence nécessaire" [Hughes and Cresswell 96], p. 314.

On notera que $x = y \supset \Box(x = y)$ combiné à l'axiome B, c'est-à-dire $(P \supset \Box\Diamond P)$ permet de dériver $x \neq y \supset \Box(x \neq y)$.

Preuve. Laissée au lecteur.

9.6.1. *Un problème ouvert en logique modale quantifiée avec identité*

Il est surprenant qu'un principe aussi général que $x = y \supset \Box(x = y)$ ait un rapport avec une propriété particulière de la relation d'accessibilité, la symétrie, qui correspond à l'axiome B. L'élucidation de ce lien inattendu reste un problème ouvert de la logique modale quantifiée. Un autre problème du même genre a été évoqué plus haut (§ 9.1.9), celui du rapport entre ce même axiome B et la constance des domaines.

9.7. La logique des termes singuliers en logique modale quantifiée

9.7.1. *Le manque d'expressivité du formalisme standard*

Jusqu'à présent, les seuls termes singuliers reconnus dans le langage de la logique modale quantifiée avec identité étaient les variables d'individus $(x, y, z, \ldots)$ et les paramètres $(a_1, b_{11}, \ldots)$. Pour augmenter l'expressivité

du langage formel, il est nécessaire d'ajouter des constantes telles que les descriptions définies comme "le tel et tel ... " et les noms propres.

On a vu plus haut l'importance de la distinction *de re – de dicto*. L'énoncé "nécessairement, il y a un premier de classe" a d'autres conditions de vérité que "il y a quelqu'un qui est nécessairement premier de classe". Cette différence sémantique se reflète syntaxiquement dans l'ordre de succession du quantificateur et de l'opérateur modal.

La distinction *de re – de dicto* s'applique aussi à des énoncés contenant la combinaison d'un opérateur modal et d'un terme singulier comme en témoignent les deux exemples ci-dessous.

(1) Un jour viendra où le président des Etats-Unis ne sera plus président des Etats-Unis.

(2) Je croyais que vous étiez plus grand que vous n'êtes.

Dans ces deux énoncés, la modalité est interprétée dans le sens *de re*. Pour faire ressortir le contraste entre les deux usages de la modalité, on fait usage d'une paraphrase. Bornons-nous au premier exemple. Dans l'interprétation *de re*, le premier énoncé se réécrit :

(3) Du président (actuel) des Etats-Unis, il sera vrai dans le futur qu'il n'est pas président des Etats-Unis.

Dans l'interprétation *de dicto*, le premier énoncé se réécrit :

(4) Il arrivera un moment où le président des Etats-Unis $\neq$ le président des Etats-Unis.

Si nous utilisons le prédicat à une place "P ①" pour représenter formellement le syntagme verbal "est président des Etats-Unis" et si nous utilisons le terme singulier "$\iota x P x$" (l'x qui est président des Etats-Unis) pour représenter formellement le syntagme nominal "le président des Etats-Unis", nous sommes incapables de refléter dans le formalisme la distinction entre la lecture *de re* et la lecture *de dicto*. En effet, les deux lectures se représentent formellement par la même formule :

(5) $F\neg P(\iota x P x)$.

9.7.2. *L'introduction des abstraits de prédicats*

Pour combler cette lacune du formalisme et *lever l'ambiguïté* de (5), Fitting et Mendelsohn enrichissent leur langage au moyen d'*abstraits de prédicats* de la forme $<\lambda x.Px>$ qu'on peut lire "la propriété d'être P".

Cet enrichissement du lexique logique s'accompagne d'un enrichissement de la grammaire logique, c'est-à-dire des règles de formation. En particulier, on aura besoin de la règle suivante :

Si $<\lambda x.Px>$ est une abstrait de prédicat et si t est un terme, l'expression $<\lambda x.Px>\ t$ est une formule.

La formule résultante peut se lire "l'objet désigné par t a la propriété exprimée par l'abstrait de prédicat $<\lambda x.Px>$".

L'avantage de cet enrichissement syntaxique réside dans le fait qu'il est désormais possible de représenter la distinction entre la lecture *de re* et la lecture *de dicto* des opérateurs modaux au moyen d'une *différence de portée* de ceux-ci.

L'énoncé (3) sera représenté par :

(6) $<F\ \neg(\lambda x.Px)>\ \iota xPx.$

L'énoncé (4) le sera par :

(7) $F\ <\neg(\lambda x.Px)>\ \iota xPx.$

9.7.3. *La syntaxe des abstraits de prédicats*

Nous avons énoncé une version sommaire de la règle de prédication (combinaison d'un terme t et d'un abstrait de prédicat). Une formulation rigoureuse exige la prise en compte des variables libres. Voici la définition complète [Fitting and Mendelsohn 98], p. 196-197.

Définition 9.2.

(A) Si B est une formule et v est une variable, $<\lambda v.B>$ est un *abstrait de prédicat*; les occurrences de variables libres dans $<\lambda v.B>$ sont celles de B, exception faite pour les occurrences de v.

(B) Si $<\lambda v.B>$ est un abstrait de prédicat et si t est un terme, $<\lambda v.B>\ (t)$ est une formule; les occurrences de variables libres dans $<\lambda v.B>\ (t)$ sont celles de $<\lambda v.B>$ plus toutes les occurrences de variables dans t.

Par exemple, dans :

(8) $(\forall y)\ <\lambda x.Px>\ (f(x,y))$,

qui formalise :

(9) Tous les y dont x est la mère (f) sont passagers (P),

seule la dernière occurrence de x est libre.

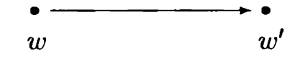
9.7.4. *La sémantique des termes et de la prédication à l'aide d'abstraits de prédicats*

Si on introduit des termes (constantes et symboles de fonction) et un nouveau type de prédication dans la syntaxe, il faut enrichir également la sémantique [Fitting and Mendelsohn 98], p. 198-199.

Soit le modèle $\mathcal{M} =<W, R, D, \mathcal{I}>$ où $w \in W$ et où g est une assignation d'un membre de D (le domaine de W) à la variable x; on associe à chaque terme t une valeur désignée par $(g * \mathcal{I})\,(t, w)$, ce qui se lit : "l'interprétation de t relativement au monde w pour l'assignation g et l'interprétation $\mathcal{I}$".

Montrons que :

$$\Diamond <\lambda x.Px> \ (c) \not\supset <\lambda x.\Diamond Px> \ (c).$$



$$D(w) = \{a, b\} \quad , \quad D(w') = \{a, b\}$$
$$\mathcal{I}(P, w) = \{a\} \quad , \quad \mathcal{I}(P, w') = \{b\}$$
$$\mathcal{I}(c, w) = a \quad , \quad \mathcal{I}(c, w') = b$$

$$\mathcal{M}, w \models_g \Diamond <\lambda x.Px> \ (c) \Leftrightarrow \mathcal{M}, w' \models_g <\lambda x.Px> \ (c);$$

$$\mathcal{M} \models_{w',g} <\lambda x.Px> \ (c) \Leftrightarrow \mathcal{M} \models_{w',g'} Px;$$

pour $g'(x) = (g * \mathcal{I})\ (c, w') = b$ et $b \in \{b\}$ donc :

$$\mathcal{M} \models_{w,g} \Diamond <\lambda x.Px> \ (c).$$

Considérons à présent $<\lambda x.\Diamond Px> \ (c)$.

$$\mathcal{M} \models_{w,g} \Diamond <\lambda x.Px> \ (c) \Leftrightarrow \mathcal{M} \models_{w,g'} \Diamond P(x);$$

où $g'(x) = (g * \mathcal{I})\ (c, w) = a$;

$$\mathcal{M} \models_{w,g'} \Diamond P(x) \Leftrightarrow \mathcal{M} \models_{w',g'} P(x).$$

On a $g'(x) = a$ et $\mathcal{I}(P, w') = \{b\}$. Comme $a \notin \{b\}$,

$$\mathcal{M} \not\models_{w,g} <\lambda x.\Diamond Px> \ (c)$$

Exercice. Démontrer :

$$<\lambda x.\Diamond Px> \ (c) \not\supset \Diamond <\lambda x.Px> \ (c).$$

L'énoncé et la réponse sont dans [Fitting and Mendelsohn 98].

9.7.5. *Les désignateurs rigides*

R. Barcan Marcus et S. Kripke ont introduit le concept de désignateur rigide dont il a été question au paragraphe 8.3.1. Rappelons que les désignateurs rigides désignent le même objet dans tous les mondes possibles du modèle dans lesquels ils existent. Les noms propres sont généralement tenus pour rigides. Certaines descriptions définies sont également des désignateurs rigides.

La description définie "le président des Etats-Unis en 1865" est un désignateur rigide, contrairement à la description définie "le président des Etats-Unis". On notera ici que pour les désignateurs rigides, la différence entre lecture *de re* et lecture *de dicto* de l'opérateur modal disparaît [Fitting and Mendelsohn 98], p. 217 :

$$<\lambda x.\Box Bx> \ (c) \equiv \Box <\lambda x.Bx> \ (c).$$

Pour une critique de la théorie de la référence des auteurs précités, on consultera [Hintikka and Sandu 95], p. 245-283.

9.7.6. *Identité nécessaire et coréférentialité*

Tant que notre lexique ne comporte que des variables, nous pouvons admettre :

$$x = y \supset \Box(x = y).$$

Ce n'est plus le cas lorsque notre lexique comporte des constantes singulières. Reprenons l'exemple célèbre de Frege. Représentons "l'étoile du matin" par $\iota x M x$, "l'étoile du soir" par $\iota x S x$ et interprétons $\Box$ par "Les anciens savaient que", il est clair que :

$$\iota x M x = \iota x S x \not\supset \Box(\iota x M x = \iota x S x).$$

Fitting et Mendelsohn distinguent l'*identité* entre objet (représentée par "=") de la *coréférentialité* entre objets désignés par des termes individuels (représentée par "$\approx$").

L'interprétation de $\approx$ est donnée par la règle d'évaluation suivante [Fitting and Mendelsohn 98], p. 205 :

$$\mathcal{M} \models_{w,g} t_1 \approx t_2 \Leftrightarrow (g * \mathcal{I})\,(t_1, w) = (g * \mathcal{I})\ ,(t_2, w).$$

Ils font observer que :

$$t_1 \approx t_2 \not\supset \Box(t_1 \approx t_2).$$

9.8. L'argument du lance-pierres

9.8.1. *Transparence et opacité*

Définition 9.3. Appelons *position référentiellement transparente* d'un terme une position qui obéit au principe de substitutivité de l'identité (S.I.).

Remarque. Nous ignorons temporairement la distinction entre identité et indiscernabilité.

Exemple. Il est vrai que Phosphorus est une planète.
Phosphorus = Hesperus.
Donc il est vrai que Hesperus est une planète.
Les termes soulignés sont en position référentiellement transparente.

Définition 9.4. Appelons *position référentiellement opaque* d'un terme une position qui n'obéit pas au principe de substitutivité. La position d'un terme dans une citation est le paradigme de la position référentiellement opaque.

Exemple. "Phosphorus" a dix lettres.
Phosphorus = Hesperus.
"Hesperus" a dix lettres.
Les termes soulignés sont en position référentiellement opaque. Comme le principe de substitutivité, la règle de généralisation existentielle est inapplicable aux termes en position référentiellement opaque.

Exemple. Jean croit vraie la phrase (fermée) "Cicéron a dénoncé Catilina".
$\exists x$ (Jean croit vraie la phrase (ouverte) "x a dénoncé Catilina").
La conclusion est mal formée. L'expression entre guillemets doit être vue comme un tout indivisible dans lequel x n'apparaît pas plus à titre de variable liée qu'il ne le fait dans :

$$\exists x \text{ (c boit du } x\text{érès).}$$

9.8.2. *Contextes intensionnels*

Définition 9.5. Appelons *contexte intensionnel* un contexte $\Phi(A)$ dans lequel les propositions *logiquement équivalentes* sont interchangeables *salva veritate*. On exprimera l'équivalence logique par le connecteur $\longleftrightarrow$ qui diffère du connecteur $\equiv$ exprimant l'équivalence matérielle.

Exemple.

$$\Box(p \supset p);$$

$$(p \supset p) \longleftrightarrow (q \supset q);$$

$$\text{donc } \Box(q \supset q).$$

Dans un contexte extensionnel, il suffit que les propositions soient *matériellement équivalentes* (c'est-à-dire qu'elles aient la même valeur de vérité) pour qu'on puisse les échanger.

Exemple. Il est vrai que César a passé le Rubicon.
César a passé le Rubicon $\equiv$ Brutus a tué César.
Donc il est vrai que Brutus a tué César.

Définition 9.6. Appelons principe d'échange des propositions logiquement équivalentes, principe noté *S.E.* par plusieurs auteurs, le principe autorisant de remplacer dans un contexte propositionnel $\Phi(p)$ une proposition simple ou complexe p par une proposition q logiquement équivalente.

9.8.3. *L'anéantissement des contextes intensionnels*

Plusieurs auteurs (Church, Quine, Davidson) ont utilisé un argument appelé aujourd'hui "argument du lance-pierres (*Slingshot Argument*)", qui vise à prouver que si l'on combine S.I. et S.E., on parvient à démontrer que les contextes intensionnels s'effondrent (comme le géant Goliath frappé avec la pierre lancée par la fronde de David).

Si l'argument est valide, il n'y a pas place, entre les contextes extensionnels (vérifonctionnels) et les contextes opaques, pour des contextes intermédiaires, les contextes intensionnels.

Sur l'argument du *slingshot*, on consultera l'article de M. Mutombo [Mutombo 95], p. 27-43.

Stephen Read [Read 93] a critiqué plusieurs versions de l'argument d'une manière détaillée. On examinera ici la version 1 dans l'article de Read et sa réfutation.

9.8.4. *Une version stylisée de l'argument du lance-pierres*

Soient p et q deux propositions contingentes vraies, telles que *César a passé le Rubicon* et *Brutus a tué César*. Soit l'expression :

(1) $\iota x(x = a \wedge p) = a \longleftrightarrow p$,

qui se lit : "il est logiquement vrai que si p est vrai, alors l'objet qui est a, pourvu que p soit vrai, est a, et réciproquement"; soit encore l'expression :

(2) $\iota x(x = a \wedge q) = a \longleftrightarrow q$.

Par *modus ponens*, puisque p et q sont vrais, on obtient :

(3) $\iota x(x = a \wedge p) = a$;

(4) $\iota x(x = a \wedge q) = a$.

Comme deux quantités égales à une même troisième sont égales entre elles, on déduit :

(5) $\iota x(x = a \wedge p) = \iota x(x = a \wedge q)$.

Soit un contexte propositionnel *quelconque* :

(6) $\Phi(p)$.

Remplaçons p par $\iota x(x = a \wedge p) = a$ en vertu de *S.E.*, nous obtenons :

(7) $\Phi(\iota x(x = a \wedge p) = a)$.

Remplaçons $\iota x(x = a \wedge p) = a$ par $\iota x(x = a \wedge q) = a$ en vertu de *S.I.*, nous obtenons :

(8) $\Phi(\iota x(x = a \wedge q) = a)$.

Remplaçons $\iota x(x = a \wedge q) = a$ par q, par *S.E.* et (8), nous obtenons :

(9) $\Phi(q)$.

La combinaison de *S.E.* et de *S.I.* conduit donc, si le *slingshot* est valide, à légitimer le principe d'échange des propositions matériellement équivalentes dans tous les contextes propositionnels y compris les contextes modaux comme le montre l'exemple suivant :

(10) □(Brutus a tué César ou Brutus n'a pas tué César),

(11) Brutus a tué César ou Brutus n'a pas tué César ≡ César a passé le Rubicon,

(12) □(César a passé le Rubicon).

Il est clair que si "César a passé le Rubicon" est une proposition nécessaire, la distinction entre le nécessaire et le contingent s'effondre.

9.8.5. *Réfutation du lance-pierres*

Read conteste l'argument du lance-pierres. La description définie qui figure dans (1) – et aussi celle qui figure dans (2) d'ailleurs – peut avoir une *portée large* ou une *portée étroite*. Elle est ambiguë.

La portée large dans laquelle la description a comme portée $x = a \longleftrightarrow p$ se représente formellement par :

(1') $[\iota x(x = a \wedge p)](x = a \longleftrightarrow p)$.

La portée étroite dans laquelle la description a comme portée p se représente par :

(1") $([\iota x(x = a \wedge p)]x = a) \longleftrightarrow p$.

Dans la première lecture, la ligne 1 est fausse. Bien que l'argument soit valide, il ne prouve pas sa conclusion puisqu'il contient une prémisse fausse.

Comme le note Read, la proposition (1') "asserte l'existence d'un objet qui est a ssi p est vrai et quoique dans le cas où p est vrai, il existe bel et bien un tel objet (à savoir a), le fait qu'il soit a n'est pas logiquement équivalent à p" [Read 93], p. 203. En effet, $a = a$ est une vérité nécessaire et p est une vérité contingente. Ces deux propositions ne peuvent donc être logiquement équivalentes. Donc (1') est fausse.

En revanche, la proposition (1") est bien logiquement vraie, mais cette fois-ci, c'est l'argument qui cesse d'être valide : "[la règle] S.I. qui autorise le remplacement de termes singuliers authentiques ne peut s'appliquer qu'à des descriptions définies qui ont une portée large" [Read 93], p. 204. Or, ici, la description définie a une portée étroite.

9.8.6. *La réhabilitation de la logique modale quantifiée*

Dans la conclusion de son article réfutant l'argument du lance-pierres, Read écrit : "La logique intensionnelle dépend de contextes $\Phi(p)$ qui ne sont pas opaques au point que p ne puisse pas être remplacé et ... ne soit pas vu du

tout, et qui ne sont pas transparents au point qu'ils puissent être remplacés par n'importe quel énoncé ayant la même valeur de vérité" [Read 93], p. 215. La conclusion s'achève sur la déclaration suivante : "La logique intensionnelle sort indemne de l'argument du lance-pierres".

On doit reconnaître que la conclusion qui découle de l'argument du lance-pierres est dévastatrice et ruine la logique modale quantifiée. L'exemple suivant permet de mesurer l'impact de la réduction des *contextes intensionnels* à des *contextes opaques.*

Prenons un énoncé mixte dont une partie figure dans un contexte intensionnel (l'attitude propositionnelle exprimée par "c sait que") et une autre partie figure dans un contexte extensionnel (la conjonction) :

> "c sait que b parle à l'x qui est F et b parle à l'x qui est F".

Si "c sait que" induit un contexte opaque, l'énoncé ci-dessus à la forme :

> "c sait qu'est vraie la phrase 'b parle à l'x qui est F' et b parle à l'x qui est F".

Il est dès lors impossible non seulement d'appliquer le *principe de substitutivité* aux occurrences de "b" et de "l'x qui est F" dans la citation, mais même d'appliquer la *règle de généralisation existentielle.* Ce n'est plus seulement la logique des termes singuliers, c'est la logique du premier ordre tout entière qui est mise hors jeu.

On ne peut plus rendre compte de raisonnements apparemment impeccables comme le raisonnement suivant :

> "c sait que b parle à l'x qui est F et b parle à l'x qui est F, donc il existe quelqu'un tel que c sait que b lui parle et b lui parle".

Il existe des alternatives à la réduction de l'intensionnel à l'opaque. Celle de Hintikka que nous allons exposer nous paraît la meilleure.

Hintikka et Sandu [Hintikka and Sandu 95], p. 252, tiennent la généralisation existentielle pour valide à condition qu'elle soit renforcée par des prémisses ajoutées :

$$\frac{\Box S(b) \quad \exists x\, \Box(b = x)}{\exists x\, \Box Sx};$$

$$\frac{[K_a]S(b) \quad \exists x\, [K_a](b = x)}{\exists x\, [K_a]Sx}.$$

Cette solution qui vaut pour la logique modale quantifiée en général avait déjà été présentée pour la logique épistémique par Hintikka [Hintikka 69], p. 133, [Gochet 75].

Hintikka énumère trois types de prémisses ajoutées :

(1) $\exists x\, (x = a)$: a existe;

(2) $\exists x\, K_c(x = a)$: c sait qui est a;

(3) $\exists x\, (x = a \wedge K_c x = a)$: a existe et c sait qui est a.

Par exemple, pour traiter l'exemple de la proposition hybride évoquée plus haut, on aura besoin d'une instance (3') de la prémisse ajoutée de type (3) :

$$[K_c]P(b, \iota x Fx) \wedge P(b, \iota x Fx);$$

$$(3') \quad \frac{\exists y\,[y = \iota x Fx \wedge [K_c](y = \iota x Fx)]}{\exists y\,([K_c]P(b,y) \wedge P(b,y))}.$$

La logique de Hintikka est une logique libre. La généralisation existentielle, quand elle n'est pas renforcée par une prémisse de type (1), ne véhicule pas de présupposition d'existence. Elle véhicule seulement une présupposition d'unicité. Pour que $\exists x\,[K_a](b = x)$ ait un sens, il faut et il suffit que b désigne le même individu "dans tous les scénarios compatibles avec ce que a sait" [Hintikka and Sandu 95], p. 251.

Exercices.

1. Quelles prémisses auxiliaires faut-il introduire pour légitimer l'inférence suivante qui comporte des *opérateurs épistémiques itérés* ("sait que", "croit que") ?

$$\frac{[K_a][B_c]S(d)}{\exists x\,([K_a][B_c]S(x))}.$$

2. Construisez un exemple montrant pourquoi la règle de nécessité est contre-intuitive en logique épistémique [Gochet 95], p. 281-294.

9.8.7. *Logique modale quantifiée et déduction naturelle*

G. Forbes [Forbes 94] a présenté une procédure de preuve en déduction naturelle pour $S5$ quantifié. Cette logique est, comme celle de Hintikka, une logique libre. D. Basin, S. Matthew et L. Vigano ([Basin et al. 97], [Basin et al. 98]) ont développé une procédure générale de déduction naturelle dans le cadre des *systèmes déductifs étiquetés* de D. Gabbay [Gabbay 96]. Elle est complète pour les sémantiques de K, D, T, B, $S4$, $S5$ quantifiés avec domaines croissants, domaines constants et même domaines décroissants.

9.8.8. *Implémentation de la logique modale quantifiée*

Une méthode de preuve par séquents avec *algorithme d'unification* pour les indices de monde a été présentée par P. Jackson et H. Reichgelt dans le chapitre 8 intitulé "A General Proof Method for Modal Predicate Logic" [Jackson et al., 89]. La procédure de D. Basin, S. Matthew et L. Vigano peut être exécutée par les démonstrateurs automatiques *Isabelle* et *LF* d'Edimbourg.

Chapitre 10
Introduction à l'intelligence artificielle

La logique classique et les logiques non classiques permettent de représenter la connaissance. Nous introduisons à présent une méthode logique permettant de manipuler la connaissance, c'est-à-dire de mettre en œuvre l'intelligence artificielle, notion qu'il convient de clarifier d'abord. Une première difficulté est que, contrairement à la plupart des domaines de science pure ou appliquée, l'intelligence artificielle n'a pas reçu de définition largement acceptée. Même dans un cas concret, un problème de programmation par exemple, il n'est pas commode de décider si on se trouve ou non dans le cadre de l'intelligence artificielle. Un moyen pragmatique de contourner cet écueil consiste à éviter de définir le concept, en se contentant d'en illustrer divers cas particuliers. Outre la frustration intellectuelle que cette omission peut engendrer, cette solution n'est pas applicable ici. Les exemples classiques reconnus de l'intelligence artificielle sont généralement complexes, et leur exposition sortirait du cadre d'un ouvrage dont l'objet n'est pas l'intelligence artificielle mais plutôt les éléments de logique qui y interviennent. La seconde difficulté que nous rencontrerons sera donc d'illustrer notre propos au moyen d'exemples très simples, voire trop simples, mais permettant quand même de mettre en évidence les spécificités de l'intelligence artificielle par rapport aux autres domaines d'application de la logique, notamment la programmation classique. Traditionnellement, l'intelligence artificielle est subdivisée en de nombreux sous-domaines, dont la plupart sont très spécialisés. Nous n'en évoquerons que quelques-uns, qui ne requièrent pas au départ de connaissance extra-logique particulière.

10.1. Une définition naturelle de l'intelligence artificielle

L'intelligence artificielle vise à simuler sur des systèmes artificiels[1] *des comportements "intelligents", typiques des êtres humains.*

A priori, cette définition ne semble pas devoir donner lieu à controverse. Divers exemples viennent spontanément à l'esprit; citons notamment le jeu d'échec, l'assistance au diagnostic médical, la traduction d'une langue dans une autre, la vision des appareils robotisés, etc. Néanmoins, cette définition suscite plus de questions qu'elle n'apporte de réponses; nous envisageons certaines de ces questions dans la suite. Signalons déjà que la notion de comportement intelligent est ambiguë; dans une moindre mesure, la notion de simulation peut aussi être interprétée de diverses manières. On peut aussi s'interroger sur le rôle privilégié accordé à l'ordinateur. L'intelligence artificielle est-elle une branche de l'informatique ? Enfin, la définition proposée est peu précise et non formelle, ce qui diminue son utilité concrète.

10.1.1. *Qu'est-ce qu'un comportement intelligent ?*

Nous n'entrerons pas ici dans un débat de psychologues et nous considérerons que les activités et comportements typiques de l'être humain sont intelligents. Par exemple, le fait de se déplacer, celui de parler, celui de regarder, constituent des comportements potentiellement intelligents. L'intelligence ne paraît pas résider dans l'aptitude à se mouvoir, à émettre des sons et à percevoir formes et couleurs, mais plutôt à contrôler et à structurer ces aptitudes. Marcher — intelligemment — implique notamment la possibilité de trouver son chemin, parler sous-entend la capacité de prendre part à une conversation et la vision suppose l'aptitude à interpréter les formes perçues en images signifiantes, en symboles, en lettres, mots et phrases dans le cas de la lecture, etc. Tous ces comportements impliquent une action, une intervention physique, mais c'est surtout le contrôle de cette action qui traduit l'intelligence. Une voiture permet le déplacement mais l'intelligence est dans le chef de son conducteur, qui contrôle ce déplacement. Dans le même ordre d'idée, même si les mains, les pieds et les yeux jouent un rôle important dans la conduite automobile, la partie intelligente de cette activité est le fait du cerveau, qui contrôle les membres et interprète les sensations visuelles transmises par la rétine.

La capacité de contrôle peut exister sans qu'il y ait intelligence. Un transistor contrôle un courant électrique, un thermostat contrôle une température et une écluse contrôle le flux d'une rivière, mais l'absence de complexité, le fait qu'un seul paramètre soit contrôlable, empêche ces objets d'être considérés comme des entités intelligentes. La complexité semble inséparable de l'intelligence, ce qui suggère la "définition" suivante :

[1] principalement des ordinateurs, ou des entités contrôlées par ordinateur.

Un comportement intelligent est une action contrôlée complexe.

On peut naturellement opposer de nombreuses objections à cette affirmation. La plus importante est peut-être son caractère réducteur : il n'est pas réaliste de vouloir traduire en une courte phrase un concept que l'on devine extrêmement sophistiqué. Dans notre contexte, cela n'est pas gênant car l'intelligence artificielle, telle que nous pouvons l'appréhender au travers de ses réalisations les plus marquantes, est fortement réductrice; nous reviendrons plus loin sur ce point. Une objection plus pertinente ici est que la complexité d'un comportement est difficile à apprécier et semble impossible à mesurer. En outre, il faut éviter de confondre la complexité d'un comportement et la complexité du résultat auquel ce comportement conduit. Par exemple, une photographie d'un paysage peut être aussi complexe qu'un tableau représentant ce paysage mais, en général, l'acte de photographier est moins complexe que la réalisation d'un tableau.

Cette dernière assertion mérite quelques justifications, qui permettront aussi d'éclaircir la notion de complexité. Dans la plupart des cas, le photographe pourra en peu de phrases décrire le mode opératoire permettant à une autre personne n'ayant pas vu la photo originale de prendre une autre photo très ressemblante à celle-ci. Il suffit de connaître le lieu, l'heure, les conditions météorologiques, l'angle de prise de vue, le type de pellicule et d'appareil utilisé, le type d'objectif, l'ouverture et la vitesse. Par contre, aussi longues que soient les indications données par le peintre, le copiste ne pourra reproduire fidèlement un tableau qu'il n'a pas vu. La longueur des instructions est une mesure d'une certaine sorte de complexité, celle qui nous intéressera ici. Nous imaginons facilement que la conception et la réalisation d'un robot peintre de tableau serait plus ardue que celle d'un robot photographe.[2] Chaque comportement, même le plus simple, comporte une complexité intrinsèque. Cela s'oppose quelque peu à une intuition selon laquelle peu de comportements seraient réellement "intelligents". Peut-être y a-t-il là une mauvaise appréhension du concept d'intelligence plutôt qu'un défaut dans la notion de complexité. L'intelligence est un concept très anthropomorphe et nous avons tendance à qualifier d'intelligent un comportement non seulement complexe, mais aussi —et surtout— mal compris. Qui pourrait expliquer pourquoi les tableaux de Claude Monet sont "bons" ? On pourrait étudier plus avant, en termes plus techniques, cette dualité intelligence / complexité mais les indications déjà données suffisent à montrer l'existence d'un lien étroit entre les deux notions. Ce lien nous sera utile car il est a priori moins simple d'estimer un niveau d'intelligence que d'estimer, voire même de quantifier un niveau de complexité.

[2]Notre propos n'est pas de prétendre à une supériorité de l'art du peintre sur celui du photographe. Cependant, la créativité de ce dernier consiste surtout à choisir son sujet et à déterminer comment il va le photographier; l'acte même de prendre la photo requiert surtout une bonne compétence technique.

A cet égard, un autre concept intéressant est celui d'apprentissage. On considère souvent qu'un comportement devant faire l'objet d'un apprentissage est intelligent; la longueur et les difficultés de l'apprentissage peuvent dans une certaine mesure se quantifier, ce qui donne une indication utile sur le degré d'intelligence requis par le comportement appris.

10.1.2. *Simuler un comportement intelligent*

Le praticien de l'intelligence artificielle ne vise pas à créer un comportement intelligent, mais seulement à le simuler. Un programme de jeu d'échecs, par exemple, doit produire des coups utiles, susceptibles de conduire à la victoire, mais il n'est pas requis qu'il "réfléchisse" ou qu'il "raisonne". Si une étape de la production du coup à jouer peut se ramener à une recherche systématique et exhaustive dans une table de coups possibles, ce n'en est que mieux; l'informaticien préfère une solution simple à une solution compliquée ... à condition naturellement que la solution simple soit aussi efficace.

Il convient de s'interroger brièvement ici sur la notion de simulation. Un point crucial est qu'une entité simulante doit ressembler à son modèle, mais le degré de similitude requis est extrêmement variable. Bien plus, pousser la ressemblance trop loin est contre-productif. Un exemple célèbre est celui des débuts de l'aviation. La machine volante de Louis Blériot était nettement supérieure à celle de Clément Ader, parce que ce dernier avait cru devoir imiter trop fidèlement le vol biologique.[3] L'évolution ultérieure de l'aviation a confirmé qu'imiter les techniques de vol des insectes, oiseaux et mammifères volants n'était pas indiqué. Observons d'ailleurs que limiter la ressemblance entre un phénomène réel et le modèle physico-mathématique permettant de l'étudier est une technique ancienne. Le physicien peut étudier avec profit les trajectoires des astres en assimilant ceux-ci à des points matériels sans dimension, dont le seul paramètre est la masse. Cela suffit notamment à déduire les lois de Kepler des principes généraux généraux de la mécanique. La ressemblance entre la Terre et le Soleil d'une part, et deux points matériels de paramètres adéquats d'autre part, est pourtant très ténue. A l'opposé, une copie d'un tableau de Monet, même très ressemblante, ne vaudra jamais qu'une petite fraction du prix de l'original. La ressemblance n'est qu'un aspect de l'adéquation entre une entité simulante et l'entité simulée.

Garder ceci à l'esprit permet d'éviter le développement d'une vue étriquée de l'intelligence artificielle. Un programme de jeu d'échecs est une réalisation de l'intelligence artificielle, dont les mérites doivent s'évaluer sur base des performances; le fait que ce programme simule plus ou moins fidèlement les moyens intellectuels mis en œuvre par un joueur humain est non pertinent: seul le résultat compte. Il est d'ailleurs plus prometteur d'essayer de simuler

[3] L'aéroplane de Clément Ader avait des ailes profilées comme celles d'une chauve-souris.

un comportement intelligent quand la qualité de ce comportement peut être évaluée avec une certaine précision. Cela signifie que la plupart du temps on tentera de simuler des comportements très particuliers, dans des contextes restreints. Le célèbre programme "Deep Blue", qui a remporté un match d'échecs contre le champion G. Kasparov, ne ressemble à ce dernier que sur un seul point, la compétence au jeu d'échecs. Même selon ce point de vue restrictif, la ressemblance est superficielle. Les mécanismes de recherche opérationnelle mis en œuvre par Deep Blue sont éloignés des modes de raisonnement humain. En résumé, Deep Blue n'est pas une entité intelligente; c'est une entité qui simule remarquablement bien, mais d'un point de vue extérieur seulement, la démarche du jeu d'échecs. A ce titre, c'est incontestablement un succès de l'intelligence artificielle.

10.1.3. *Intelligence artificielle et informatique*

Un joueur d'échec artificiel est censé jouer convenablement aux échecs mais on n'attend rien d'autre de lui. Il n'est pas nécessaire qu'il sache parler ou marcher, ni même qu'il puisse lui-même mouvoir les pièces et identifier les coups joués par l'adversaire. En fait, le joueur artificiel doit seulement pouvoir communiquer, sous n'importe quelle forme, les coups à jouer; on doit aussi pouvoir lui communiquer les coups joués par l'adversaire. On reconnaît ici le caractère restrictif d'une simulation de comportement intelligent. Admettre cette restriction permet de traiter la plupart des problèmes d'intelligence artificielle par l'informatique. L'entité simulant l'intelligence est simplement un ordinateur convenablement programmé; elle ne fait que traiter de l'information.

Un autre exemple instructif est celui du simulateur de vol. La composante essentielle de cette machine sophistiquée est l'ordinateur, même si les équipements périphériques tels les capteurs et les actuateurs sont indispensables pour permettre un niveau approprié de simulation. Cette simulation reste néanmoins très infidèle, puisque le simulateur est dépourvu de la caractéristique principale de l'avion, à savoir celle de voler. Cette "lacune" n'est pas gênante puisque le simulateur de vol ne teste pas l'habileté du candidat pilote à se mouvoir rapidement en altitude, mais bien celle de réagir rapidement et correctement à un environnement de vol et aux divers incidents qui peuvent se produire. Ces exemples, et d'autres que nous verrons plus loin, nous incitent à considérer que l'intelligence artificielle est principalement une subdivision de l'informatique.

10.1.4. *Le point de vue anthropomorphe*

Le comportement intelligent est pour nous une action contrôlée complexe mais nous considérons seulement sa partie centrale, qui consiste en un traitement d'informations. Par exemple, conduire une voiture requiert une bonne perception de l'environnement, essentiellement visuelle. Les sensations visuelles

sont interprétées en informations diverses, dont la partie utile est "filtrée"[4] et transmise à la partie décisionnelle du cerveau, qui déterminera les commandes motrices transmises aux muscles, qui eux-mêmes agiront sur le volant, l'accélérateur, etc. Les aspects sensoriels et moteurs ne sont habituellement pas considérés comme faisant partie du processus intelligent au sens strict, au contraire de l'étape centrale intermédiaire. De même, l'intelligence artificielle au sens strict n'étudie pas les capteurs, analogues de nos sens, ni les actuateurs, analogues de nos muscles, mais seulement le processus par lequel les données issues des capteurs sont transformées en commandes pour les actuateurs. Tant les données que les commandes sont de l'information, ce qui nous amène à admettre que le succédané artificiel du cerveau ne peut être que l'ordinateur convenablement programmé. On peut objecter à cela que, même sous l'angle structurel, l'ordinateur et le cerveau ont peu de ressemblance mais, comme nous l'avons déjà mentionné, vouloir améliorer cette ressemblance risque d'être contre-productif.[5] D'ailleurs, le lien entre le cerveau et l'intelligence est controversé et on peut difficilement confondre le cerveau proprement dit ("brain") et l'esprit ("mind"), que l'on définit précisément comme le siège de l'intelligence. Certains spécialistes estiment que notre cerveau n'est qu'une machine plutôt stupide, qui doit être remplie d'une grande quantité d'information avant de devenir intelligente; l'analogie avec l'ordinateur qui est inutile tant qu'il n'est pas pourvu d'un système d'exploitation et de programmes d'application est assez naturelle ici. On hésite à aller plus loin et à prétendre que l'être humain ne serait pas intelligent de naissance, mais qu'il le deviendrait au fur et à mesure que son cerveau se "remplit".

Sans adopter un point de vue aussi radical, on peut admettre que l'intelligence artificielle n'est pas le fait de l'ordinateur lui-même, mais des programmes et des données que sa mémoire comporte. Notons ici que la mémoire fait partie de l'intelligence (artificielle ou non).[6] En intelligence artificielle, on s'intéressera plus aux données et aux programmes qu'aux ordinateurs eux-mêmes. Dans les chapitres suivants, on étudiera notamment PROLOG, que l'on peut voir comme un programme intelligent générique; ce programme peut être spécialisé de manière à le rendre apte à résoudre des problèmes particuliers. On étudiera aussi des techniques de démonstration automatique.

---

[4] Si on envisage de dépasser un autre véhicule, on pourra noter plus ou moins inconsciemment sa marque, sa couleur, le numéro de la plaque, etc., mais l'information utile est son mouvement, c'est-à-dire sa position et sa vitesse; on déterminera aussi si l'environnement routier se prête à un dépassement, mais on ignorera sans doute une foule de détails inutiles.

[5] Les réseaux neuronaux artificiels imitent mieux la structure cérébrale que les ordinateurs classiques mais, pour de nombreuses tâches, sont moins performants que ces derniers.

[6] Dans la vie quotidienne, on note que les gens sont plus disposés à reconnaître des défaillances de mémoire plutôt qu'un défaut d'intelligence. En fait, la mémoire ne peut être séparée de l'intelligence; les gens intelligents se plaignant de leur mémoire déplorent le plus souvent d'oublier des *faits* mais ils gardent en mémoire des structures plus élaborées comme des règles de conduite, des algorithmes, des techniques, etc.

10.1.5. *Intelligence et traitement de l'information*

L'exemple de la vision, naturelle ou artificielle, illustre bien les multiples aspects d'un comportement intelligent, c'est-à-dire d'une action contrôlée complexe. On peut considérer la vision comme un processus en deux étapes. La première est le fait de l'œil et conduit à une image rétinienne, c'est-à-dire à une très grande matrice de points lumineux, souvent appelés pixels. Vu le nombre énorme de pixels et le fait que chacun d'eux est susceptible de prendre plusieurs états, une image représente une très grande quantité d'information. Cependant, dans la plupart des cas, cette information sera "filtrée" et résumée en une faible quantité d'information utile, exprimable en une courte phrase, telle "voilà Jacques" ou "il fait beau ce matin". L'aptitude à isoler et à structurer en quelques concepts l'information pertinente noyée au sein d'une vaste quantité d'information brute, visuelle ou autre, est une composante essentielle de l'intelligence humaine, qu'il sera important de simuler dans un programme d'intelligence artificielle. Cette aptitude est d'ailleurs systématiquement mesurée par les tests d'intelligence, comme le montre le dessin de la figure 10.1. L'information brute comprise dans ces dessins est abondante; on s'en rend compte si l'on essaie de décrire méthodiquement et exhaustivement chacun de ces dessins. Par contre, si on lui demande ce qui distingue les quatre cases de gauche des quatre cases de droite, l'être humain devra "nettoyer" cette information pléthorique et abstraire de ces dessins que la partie gauche illustre le concept "3" tandis que la partie droite illustre le concept "4".

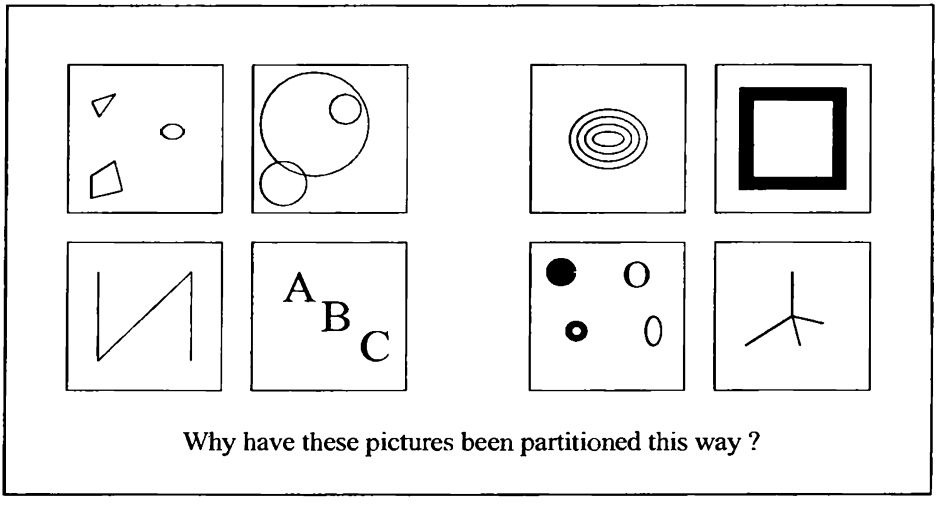


Figure 10.1 *Extraire l'information visuelle et conceptuelle pertinente*

L'esprit humain détermine facilement et rapidement que les huit dessins représentent chacun un nombre mais écrire un programme capable de détecter cela et, plus généralement, de passer avec succès un test d'intelligence de ce type est délicat.[7] Une des nombreuses aptitudes visuelles de l'esprit humain consiste en la reconnaissance des formes, en dépit de leur diversité. On peut créer un programme de reconnaissance de formes et le "nourrir" avec, d'une part, une série de photos d'arbres et, d'autre part, une série de photos de chats, et escompter que par la suite, ce programme soit capable de classer des photos (autres que celles qu'il connaît) selon qu'elles représentent un arbre ou un chat. Un premier module du programme fera de la détection de contour; il convertira chaque image (tableau de pixels) en un nombre relativement faible de segments de droite ou de courbe. Cette étape préliminaire n'est pas vraiment du domaine de l'intelligence artificielle, quoique l'on y reconnaisse un aspect "intelligent" qui est la compression, l'abstraction de l'information. Un deuxième module fera du "nettoyage", c'est-à-dire qu'il décidera, pour chaque segment, de le maintenir, de le supprimer ou de le modifier, par exemple pour le connecter à ses voisins. Un troisième module comparera le contour nettoyé à divers contours représentant des "standards" d'arbre ou de chat et décidera de classer l'image analysée dans l'une ou l'autre catégorie (ou dans aucune). Ce module utilisera diverses techniques (unification de l'image analysée et d'une image de référence par des transformations géométriques telles que les translations, rotations et homothéties, recherche de caractéristiques physiques discriminantes, etc.). L'arbre schématisé à la figure 10.2 montre une fois de plus qu'un modèle n'a pas besoin d'être fidèle pour être utile. Même si la silhouette d'arbre est très sommaire, une fraction de seconde suffit à la distinguer des trois silhouettes de chat représentées à droite de la figure. L'une des silhouettes est très différente des deux autres, parce que le chat est assis; cette différence ne trouble pas l'observateur humain, mais on conçoit aisément qu'une intelligence visuelle non négligeable soit nécessaire pour discerner les similitudes entre ces silhouettes et en tirer les conclusions adéquates.

Une autre aptitude liée à l'intelligence est la synthèse d'une information utile à partir de plusieurs informations fragmentaires et apparemment disparates. Considérons par exemple la phrase

Vincent est le fils du mari
de la fille unique de ma belle-mère.

On peut "décoder" cette phrase en

Vincent est mon fils.

[7] Ecrire un programme capable de compter des objets sur une image est relativement simple; la difficulté ne consiste pas à mettre en œuvre un concept particulier, connu à l'avance, mais à manier différents concepts et même à les créer. L'image de la figure 10.1 fait partie d'une série dont chaque élément concerne des concepts visuels distincts (objets ouverts ou fermés, groupés ou disséminés, de tailles voisines ou de tailles très différentes, etc.).

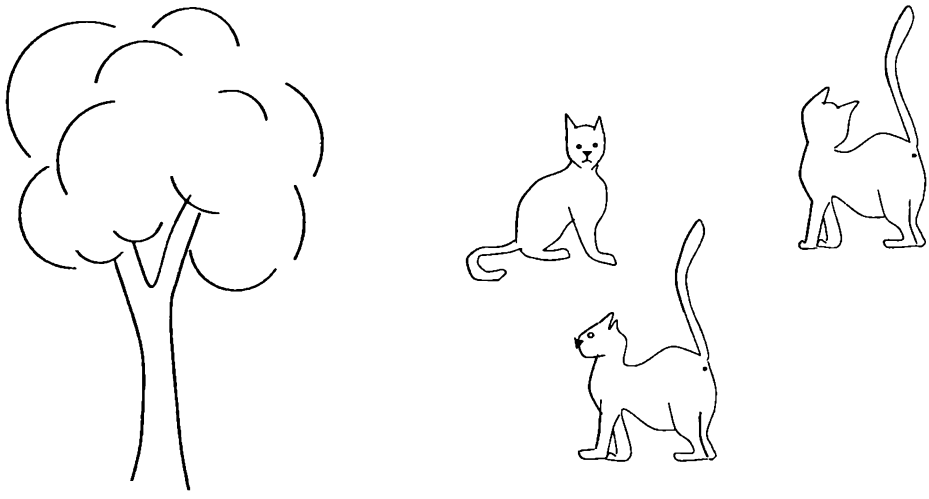


Figure 10.2 *Représentation sommaire de l'Arbre et du Chat*

Beaucoup d'énigmes peuvent se résoudre par un décodage de ce type. Par exemple, analysons le texte qui suit, se rapportant à un concours auquel trois étudiants de différentes nationalités ont pris part; chacun d'eux a son sport favori.

Michel joue au football et est mieux classé que l'Américain.
Simon est Belge et a surclassé le joueur de tennis.
Le nageur s'est classé premier.

Les questions sont :

Qui est le Français ?
Quel sport pratique Richard ?

Il existe une technique simple pour résoudre une énigme de ce type. Elle consiste à énumérer toutes les possibilités puis à les "filtrer" en fonction des indices donnés. Cette méthode, si simple qu'elle soit dans son principe, est une des composantes majeures de notre intelligence et probablement la technique la plus importante en intelligence artificielle. On voit facilement ici qu'il y a a priori $(3!)^3 = 216$ possibilités d'apparier les quatre triplets

{ *Michel, Simon, Richard* },
{ *Football, Tennis, Natation* },
{ *Americain, Belge, Français* },
{ *Premier, Deuxième, Troisième* }.

Une seule de ces possibilités vérifie les trois indices :

Premier : le Belge Simon, nageur;

Deuxième : Michel, qui est Français et joue au football;
Troisième : Richard, l'Americain, qui est joueur de tennis.

De telles énigmes sont élémentaires parce qu'elles se ramènent à la construction et à l'exploration systématique d'un espace structuré et fini de solutions potentielles. Cela n'était pas vrai pour le test visuel de QI évoqué plus haut (Figure 10.1), notamment parce qu'aucune liste exhaustive de concepts susceptibles d'être visuellement illustrés n'est disponible. En fait, l'ensemble de ces concepts n'est qu'assez vaguement défini. Cet aspect n'est d'ailleurs pas spécifique à la vision. Au contraire, dans bien des cas, la "règle du jeu" n'est pas fournie, et la démarche intelligente requise consiste à découvrir cette règle plutôt qu'à la mettre en œuvre. Que signifie par exemple la séquence:

Which Atom Issue Art Tiny Finance Cigarette
All Inside Tear Llama Ignorant Enable Ceylon

composée de mots sans suite logique apparente ?[8]

10.2. Une définition orientée de l'intelligence artificielle

Le but de l'intelligence artificielle est de simuler, au sens large, le travail de contrôle effectué par le cerveau. "Au sens large" signifie que seul le résultat de la simulation compte; les processus mis en œuvre en intelligence artificielle ne sont pas censés ressembler aux processus physiologiques dont le cerveau humain est le siège.[9] L'intelligence artificielle est donc une branche de l'étude des programmes. Etudier un programme signifie s'intéresser à la fonction de contrôle que possède l'ordinateur exécutant ce programme, mais n'implique pas que l'on se préoccupe des processus physiques, électroniques, mis en œuvre lors de l'exécution. Nous considérerons donc les ordinateurs comme des machines abstraites capables d'interpréter les programmes qui leur sont soumis, sans nous soucier de leur fonctionnement interne.

Concrètement, l'intelligence artificielle a donc pour objet l'élaboration de programmes munis d'une fonction de contrôle ou, plus généralement, d'un aspect intelligent. Nous allons préciser les notions de programme et de programme intelligent.

10.2.1. *Qu'est-ce qu'un programme ?*

Tout programme a pour rôle de contrôler le fonctionnement de l'ordinateur qui l'exécute. Un ordinateur est une machine à états et le programme induit

[8] "WhAt Is ArTiFiCiAl InTeLlIgEnCe ?"

[9] On peut aussi s'intéresser à la simulation sur machine des processus de l'intelligence naturelle, mais ces derniers ne sont que très imparfaitement connus. Ce type de simulation donne rarement des résultats techniquement intéressants. Une exception notable est constituée par les réseaux de neurones artificiels, qui ont de multiples applications. Nous ne les évoquerons pas ici, car leur fonctionnement n'est pas basé sur la logique formelle.

une exécution, c'est-à-dire une suite d'états. L'exécution correspond donc à une trajectoire dans l'espace des états et le programme contrôle cette trajectoire. Cet aspect de contrôle ne suffit pas à faire de tout ordinateur exécutant un programme quelconque une machine intelligente. Pour mieux appréhender l'aspect "intelligent" éventuellement présent dans un programme, nous considérons quelques exemples, en commençant par un contre-exemple.

L_0 : $y := 1;$
L_1 : while $x > 0$ do
L_2 : $\quad y := y * x;$
L_3 : $\quad x := x - 1;$
L_4 : $F := y.$
L_5 :

Ctrl	x	y	F
L_0	3	$\perp$	$\perp$
L_1	3	1	$\perp$
L_2	3	1	$\perp$
L_3	3	3	$\perp$
L_1	2	3	$\perp$
...		...	
L_1	0	6	$\perp$
L_4	0	6	$\perp$
L_5	0	6	6

Figure 10.3 *Un programme impératif et un exemple d'exécution*

Le programme classique de calcul de la factorielle d'un nombre naturel est représenté à la figure 10.3, avec l'une de ses exécutions. D'un point de vue opérationnel, ce programme induit une séquence d'opérations arithmétiques élémentaires qui conduit au résultat, rangé dans la variable F. Pour justifier que le programme fonctionne bien, c'est-à-dire qu'il fournit le résultat exact, on utilise la méthode des invariants (voir § II.3). Un invariant approprié est :

$$\begin{aligned}
& at\ L_0 \Rightarrow x = x_0 \\
\wedge\ & at\ L_1 \Rightarrow (0 \leq x \leq x_0 \wedge y * x! = x_0!) \\
\wedge\ & at\ L_2 \Rightarrow (0 < x \leq x_0 \wedge y * x! = x_0!) \\
\wedge\ & at\ L_3 \Rightarrow (0 < x \leq x_0 \wedge y * (x-1)! = x_0!) \\
\wedge\ & at\ L_4 \Rightarrow (x = 0 \wedge y = x_0!) \\
\wedge\ & at\ L_5 \Rightarrow (x = 0 \wedge y = x_0! \wedge F = x_0!)\,,
\end{aligned}$$

où x_0 désigne la valeur initiale de x, c'est-à-dire un entier naturel quelconque. La vérification elle-même est un processus en deux temps. On s'assure d'abord que la formule ci-dessus est bien un invariant, et ensuite que cet invariant implique le bon fonctionnement du programme. La première étape est élémentaire mais requiert une connaissance suffisante de la fonction factorielle et des opérations arithmétiques qui permettent de la définir. Par exemple, pour vérifier la validité de la transition conduisant du point de contrôle L_2 au point

de contrôle L_3, il faut disposer du théorème suivant :

Pour tous entiers x_0, x et y,
si $0 < x \leq x_0 \wedge y * x! = x_0!$ est vrai,
alors $0 < x \leq x_0 \wedge (y * x) * (x - 1)! = x_0!$ est vrai.

Ce théorème, quoique très simple, découle d'un certain nombre de vérités arithmétiques, notamment celles-ci :

$$\forall x \forall y \forall z [x * (y * z) = (x * y) * z],$$
$$\forall x [x > 0 \Rightarrow x! = x * (x - 1)!].$$

La seconde étape consiste à montrer que cet invariant (ou plutôt le programme auquel il se rapporte) induit un moyen de calcul fiable de la factorielle d'un nombre naturel quelconque x_0. Cette étape est ici très simple : l'invariant indique explicitement que tout calcul démarrant au point L_0 (en un état vérifiant $x = x_0$) et atteignant le point L_5 est tel que l'état final vérifie $F = x_0!$. Il est en outre facile de déterminer que ce calcul requiert l'exécution de $3x_0 + 3$ transitions.

Le procédé complet de calcul de la factorielle d'un entier naturel peut être structuré en trois phases comme suit.

- Les mathématiciens fournissent une définition (ou plusieurs définitions équivalentes) de la fonction factorielle reposant sur des fonctions et concepts mathématiques antérieurement définis (comme les opérations élémentaires) et sur des propriétés déjà démontrées de ces fonctions et concepts (associativité de la multiplication, par exemple).
- Le concepteur de programme sélectionne une définition se prêtant à un calcul effectif et la transforme en un programme exécutable. Le rôle de l'invariant est de convaincre le programmeur de la qualité de son travail, en reliant le programme à la connaissance mathématique qu'il met en œuvre.
- L'ordinateur, muni du logiciel adéquat (un système d'exploitation, un interpréteur ou un compilateur, ...), exécute le code; cela signifie que l'ordinateur passe, en un certain nombre d'étapes, d'un état initial comportant les données à un état final comportant les résultats.

Ce procédé requiert clairement de l'intelligence, de la part du mathématicien et aussi du programmeur. On peut défendre l'idée que le programme et l'ordinateur eux-mêmes présentent une certaine forme d'intelligence, mais elle est ici très limitée. La machine programmée ne fait qu'exécuter des instructions très détaillées et très spécifiques, dans un but bien précis. Il est néanmoins clair que l'intelligence réside essentiellement dans les deux premières phases, et très peu, ou pas du tout, dans la troisième.

Du point de vue de l'informaticien, il n'est pas nécessaire d'approfondir ce point et de décider si l'ordinateur, dont l'intervention est limitée à la troisième phase, est ou non pourvu d'une capacité ressemblant à l'intelligence; la question

intéressante est plutôt de savoir si le rôle de l'ordinateur (et du logiciel) peut être accru, réduisant d'autant le rôle des humains, impliqués dans les deux premières phases. On convertirait ainsi un ordinateur en un système (relativement) intelligent.

10.2.2. *Qu'est-ce qu'un système intelligent ?*

L'exemple très élémentaire que nous venons d'évoquer suggère une réponse, naturellement partielle, à cette question. Un système relativement intelligent serait capable, par exemple, de déduire la factorielle d'un nombre naturel directement de la connaissance mathématique concernant cette fonction, sans passer par l'intermédiaire d'un texte opérationnel, écrit par un concepteur de programme. Cette "définition" n'est pas vraiment satisfaisante puisqu'il n'y a pas une frontière claire entre la connaissance mathématique et la connaissance opérationnelle. En particulier, il n'est pas absurde de considérer que le texte

(10.1) $(x, y) := (n, 1)$; while $x > 0$ do $(x, y) := (x - 1, y * x)$; return y,

quoique typiquement opérationnel, constitue une définition mathématiquement acceptable de $n!$, même si le mathématicien préférera naturellement une définition telle que

(10.2) $0! = 1$; pour tout $n > 0$, $n! = n * (n - 1)!$

ou encore que

$$(10.3) \quad n! = \int_0^{+\infty} x^n e^{-x}\, dx\,.$$

La définition (10.2) semble plus déclarative que la définition (10.1), ce qui la rend plus attrayante pour le mathématicien, mais la différence entre les deux n'est pas très importante : la plupart des langages de programmation admettent la récursion, ce qui rend la seconde définition aussi opérationnelle que la première. La définition (10.3) est intéressante notamment parce qu'elle permet d'étendre la fonction factorielle à l'ensemble des nombres complexes, entiers négatifs exclus. D'un point de vue opérationnel, par contre, cette définition est moins intéressante parce qu'elle recourt à des concepts comme l'exponentielle et l'intégration, moins élémentaires que les opérations arithmétiques et plus difficiles à mettre en œuvre.

D'une manière générale, le mathématicien privilégie un langage spécifique, comportant les égalités, les équations, les relations, etc., alors que les langages de programmation sont basés sur des notions assez différentes, comme les itérations, les séquences, les affectations, les procédures. Les deux types de langages admettent des recouvrements partiels; par exemple, la notion mathématique de fonction est proche de la notion informatique de procédure. Un premier pas, relativement modeste, vers les systèmes intelligents, consiste à

doter la machine de l'aptitude à interpréter opérationnellement des fragments de connaissance mathématique écrits en un langage mathématique, déclaratif. Notons que le langage mathématique formel est, le plus souvent, une logique; ceci est une première explication du lien étroit existant entre la logique et l'intelligence artificielle.

On peut estimer que, dans le cas du programme de calcul de la factorielle donné plus haut, le travail de construction est réduit et qu'il n'est donc pas très utile de vouloir s'en dispenser. Remarquons néanmoins que ce travail est plus important qu'il n'y paraît, puisque s'assurer de la correction du programme requiert la construction de l'invariant — une formule relativement longue — et la validation de celui-ci. Le fait que le processus de calcul associé au programme soit le même que le processus de calcul employé par un être humain pour calculer une factorielle n'incite pas non plus à chercher une solution plus "intelligente", qui ne serait sans doute pas meilleure.

Nous pouvons admettre que tout moyen permettant de réduire le travail de conversion de la connaissance mathématique déclarative en connaissance informatique, c'est-à-dire en un programme exécutable, est un pas vers l'intelligence artificielle. En effet, la machine prend alors en charge, en partie du moins, un travail qui, accompli par l'être humain, paraît requérir une démarche intellectuelle.

Des langages fonctionnels comme Lisp, Scheme et ML permettent la mise en œuvre opérationnelle immédiate de connaissances "semi-déclaratives" d'un type particulier, à savoir les définitions récursives telle que (10.2). Des systèmes de calcul symbolique (MACSYMA et REDUCE par exemple) permettent d'exploiter de la connaissance mathématique déclarative de types plus variés, et notamment la définition intégrale (10.3). Le mécanisme d'exploitation opérationnelle de la récursion est, en principe du moins, extrêmement simple; il ne suffit donc pas à faire de Lisp un système d'intelligence artificielle dans l'acception courante du terme. En ce qui concerne REDUCE et MACSYMA, les mécanismes sont plus nombreux et plus sophistiqués.

10.2.3. *Les Tours de Hanoï*

Un autre exemple bien connu, le problème des Tours de Hanoï, peut aider à mieux appréhender le concept d'intelligence artificielle. On dispose de trois piquets, nommés A, B et C, et de N disques de tailles échelonnées, percés en leur centre de manière à pouvoir être empilés sur les piquets. Un disque plus grand ne peut jamais se trouver au-dessus d'un disque plus petit. Au départ, tous les disques se trouvent empilés sur le piquet A. Le problème consiste à construire une suite de mouvements élémentaires, impliquant chacun le déplacement d'un seul disque, conduisant à la situation où tous les disques sont empilés sur le piquet B.

La manière classique d'aborder ce problème consiste à observer d'abord qu'une solution immédiate existe dans le cas où $N = 1$. En outre, et ceci

est moins évident, on observe que, pour déplacer N disques du piquet A vers le piquet B, il faudra procéder nécessairement en trois étapes. La première consiste à transférer $N-1$ disques du piquet A au piquet C, la deuxième à transférer un disque (le plus grand) du piquet A au piquet B et la troisième à transférer $N-1$ disques du piquet C au piquet B. Cette tactique typiquement récursive est illustrée à la figure 10.4, dans le cas $N = 5$.

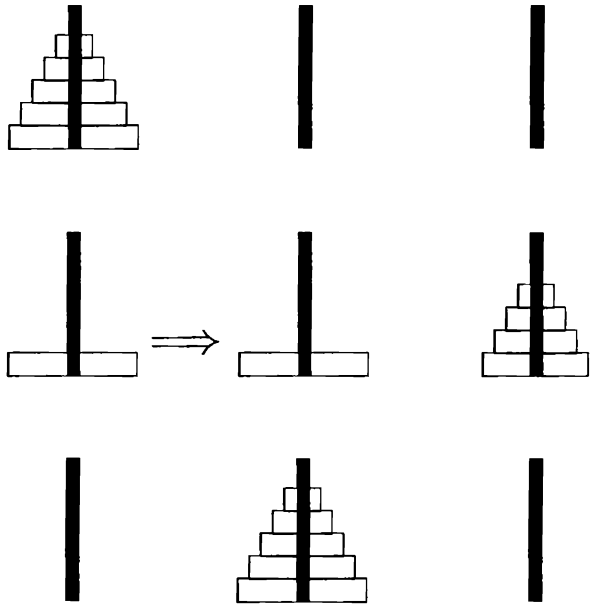


Figure 10.4 *Tours de Hanoï: approche classique*

Cette approche donne lieu au programme récursif ci-dessous:

$$\begin{aligned}\text{Towers}(1, X, Y, Z) &= [X \to Y] \\ \text{Towers}(N+1, X, Y, Z) = \quad &\text{Towers}(N, X, Z, Y) \\ &* \text{Towers}(1, X, Y, Z) \\ &* \text{Towers}(N, Z, Y, X)\end{aligned}$$

(Le symbole $*$ désigne ici l'opérateur de concaténation de listes.)

Cette manière de résoudre le problème des Tours de Hanoï n'est pas considérée habituellement comme un exemple d'intelligence artificielle parce que la machine, ou plutôt l'interpréteur du langage fonctionnel utilisé, n'a déchargé l'humain que d'une tâche modeste (en principe), à savoir l'exploitation de la récursivité. Remarquons aussi que la séquence de mouvements induits

par ce programme est optimale, c'est-à-dire aussi courte que possible. Si l'on note $\lambda(N)$ cette longueur, on a :

$$\lambda(1) = 1\,,$$
$$\lambda(N+1) = 1 + 2\,\lambda(N)\,,$$

d'où on déduit $\lambda(N) = 2^N - 1$.

Pour voir que la tâche de "dérécursivation" est modeste dans son principe, il suffit de la simuler. On considère la relation récursive comme une règle de réécriture et on convertit la liste initiale, comportant un seul mouvement composé :

$$\text{Towers}(N, X, Y, Z)$$

en la liste :

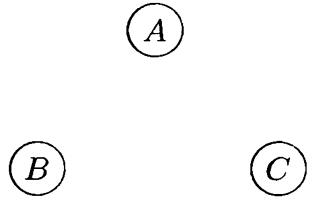
$$\text{Towers}(N-1, X, Z, Y)$$
$$\text{Towers}(1, X, Y, Z)$$
$$\text{Towers}(N-1, Z, Y, X)$$

comportant trois mouvements dont le premier et le dernier sont encore composés, sauf si $N = 2$. On réapplique le même procédé de réécriture pour obtenir

$$\text{Towers}(N-2, X, Y, Z)$$
$$\text{Towers}(1, X, Z, Y)$$
$$\text{Towers}(N-2, Y, Z, X)$$
$$\text{Towers}(1, X, Y, Z)$$
$$\text{Towers}(N-2, Z, X, Y)$$
$$\text{Towers}(1, Z, Y, X)$$
$$\text{Towers}(N-2, X, Y, Z)$$

et ainsi de suite. Après $N-1$ répétitions du procédé, on obtient la liste optimale des mouvements élémentaires.

Il est intéressant d'observer qu'une autre approche est possible, dont la mise en œuvre ne requiert pas la maîtrise du mécanisme de la récursion. Supposons que les trois piquets soient disposés en triangle, comme ci-dessous :



Le programme suivant convient, quand N est pair :

Répéter

- déplacer le petit disque dans le sens des aiguilles d'une montre;
- s'arrêter si la situation finale est atteinte;
- déplacer un autre disque, de la seule manière possible.

(Si N est impair, le sens de rotation du plus petit disque est inversé.)

La tâche de la machine est ici de simple exécution; cette approche a requis plus d'intelligence de la part du concepteur, et donc moins de la part de la machine. On peut s'en rendre compte facilement : se convaincre de l'exactitude du second programme est moins simple que pour le premier. On peut aussi considérer que la tactique récursive suggérée dans la première approche est plus facile à imaginer et (légèrement) plus difficile à mettre en œuvre que la tactique itérative de la seconde approche.

Le but ultime de l'intelligence artificielle serait de dispenser l'être humain de fournir une tactique; le système intelligent disposerait à l'avance de toutes les tactiques nécessaires ou, encore mieux, de l'aptitude lui permettant de les "concevoir" au fur et à mesure qu'elles se révéleraient nécessaires. Cette ambition est naturellement irréaliste mais, en délimitant le type de problèmes que la machine intelligente est censée résoudre, on peut arriver à des résultats intéressants. Les deux tactiques fournies pour résoudre notre exemple des Tours de Hanoï sont spécifiques à ce problème mais on peut concevoir une tactique plus générale, qui s'appliquerait à une vaste classe de problèmes.

Cette tactique générale provient d'une constatation très simple. A chaque étape, le nombre de mouvements possibles est limité. On peut donc représenter le problème des Tours de Hanoï par un arbre finitaire; la racine correspond à la situation initiale et tout nœud correspondant à une situation donnée admet pour successeurs directs les situations que l'on peut atteindre en déplaçant un seul disque. Chaque branche représente donc une suite licite de mouvements. Il suffit donc de construire l'arbre, "en largeur d'abord", pour obtenir la meilleure solution au problème. Cette tactique de construction d'arbre s'applique à une très large classe de problèmes, mais son intérêt pratique semble limité par un phénomène évident : le processus de construction de la solution est désespérément lent. Dans le cas qui nous occupe, la plupart des situations admettent trois successeurs directs, deux correspondant à un mouvement du petit disque, et un correspondant au mouvement d'un autre disque. Cela signifie que pour obtenir à coup sûr la solution optimale dans le cas de N disques, il faudrait construire un arbre ternaire de profondeur $2^N - 1$, comportant jusqu'à 3^{2^N-1} feuilles (et autant de nœuds internes). C'est naturellement impossible sauf pour les plus petites valeurs de N. On peut cependant observer que dans le cas présent, ainsi d'ailleurs que pour la plupart des problèmes similaires, l'arbre peut être sérieusement élagué. Tout d'abord, l'un des trois successeurs d'un nœud est nécessairement le prédécesseur de ce nœud et peut, de ce fait, être omis; il est en effet inutile de repasser par une situation déjà rencontrée. On peut encore réduire l'arbre

en excluant des successeurs non seulement le prédécesseur mais aussi toute situation déjà rencontrée plus haut dans l'arbre, même dans une autre branche. Ces optimisations élémentaires suffisent à rendre plus acceptable la taille de l'arbre, qui est représenté à la figure 10.5 dans le cas de trois disques. On pourrait encore réduire l'arbre en tenant compte des symétries; si on connaît la suite $S(3, A, B)$ de mouvements nécessaires pour faire passer, par exemple, trois disques du piquet A vers le piquet B, on obtiendra $S(3, C, A)$ en appliquant à $S(3, A, B)$ la substitution $(A, B, C) := (C, A, B)$.

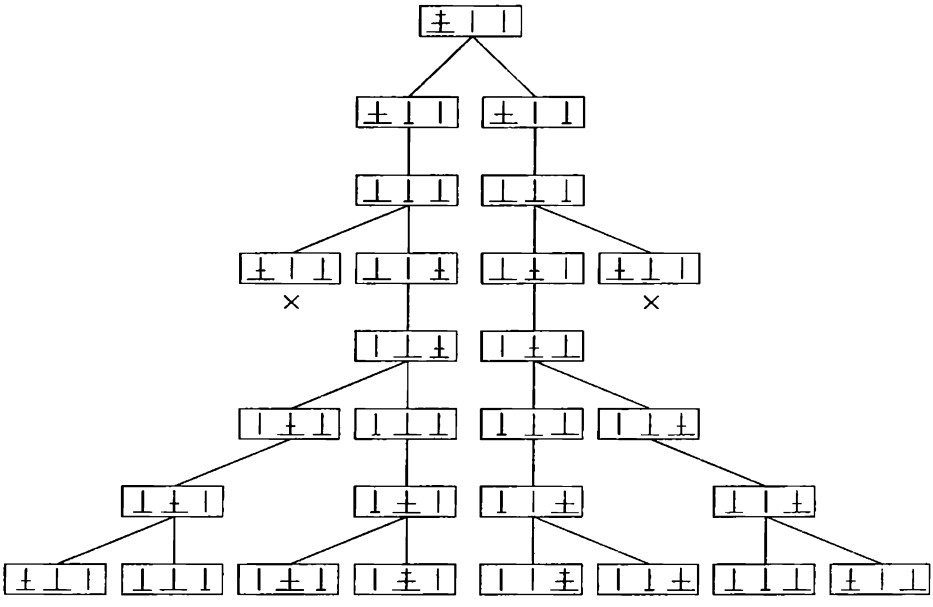


Figure 10.5 *Tours de Hanoï: recherche systématique*

Il est peu intéressant d'appliquer la recherche systématique dans le cas du problème des Tours de Hanoï parce qu'une tactique plus spécifique et aussi plus efficace est connue ... c'est-à-dire parce que la partie intelligente du problème a déjà été effectuée. En fait, on pourrait dire que ce problème *n'appartient plus* à l'intelligence artificielle. Les exemples qui suivent montreront que l'appartenance d'un problème à l'intelligence artificielle dépend aussi de la connaissance que nous avons de ce problème. En fait, une situation parfaitement connue requiert moins le recours à l'intelligence (artificielle ou non) qu'une situation mal connue. De toute manière, la frontière entre les situations requérant de l'intelligence et les autres sera naturellement floue. Un autre point important, bien illustré par le problème des Tours de Hanoï, est que, si on a le choix entre l'approche classique et l'approche intelligence artifi-

cielle, il faut choisir la première. Ceci n'est paradoxal qu'en apparence. Dans une situation où l'être humain peut réagir par réflexe, il aurait généralement tort de prendre le temps de vérifier le bien-fondé de l'action réflexe avant de l'accomplir.[10]

10.2.4. *Nim*

Le jeu de Nim met en compétition deux joueurs. Il nous intéresse parce que l'on peut y jouer en utilisant les ressources de l'intelligence, naturelle ou non, avec un certain intérêt; cependant, une tactique infaillible, ne requérant pas de raisonnement mais un simple calcul, permet de jouer de manière infaillible. Un certain nombre de tas d'allumettes sont disponibles au départ. (La configuration initiale habituelle est de quatre tas, comportant respectivement sept, cinq, trois et une allumette, respectivement.) A tour de rôle, chaque joueur enlève une ou plusieurs allumettes, prises dans un seul tas, au choix. Le gagnant est le joueur prenant la dernière allumette.

La démarche du joueur humain ne connaissant pas la tactique spécifique du jeu sera d'anticiper les réactions possibles de l'adversaire, en prévoyant une certain nombre de coups à l'avance. Cette stratégie est optimale ... à condition de pouvoir la mettre en œuvre complètement. Le nombre de situations possibles croît très vite en fonction du nombre de coups considérés. Un programme d'intelligence artificielle procédera de même, en construisant un arbre de jeu analogue à celui déjà évoqué pour le problème des Tours de Hanoï. Les coups joués par le programme viseront à explorer une branche de cet arbre, gagnante pour le programme. En pratique, si le nombre total d'allumettes disponibles au départ n'est pas très élevé, cette stratégie fonctionnera très bien; elle devient cependant impraticable s'il y a par exemple 200 allumettes réparties en dix tas.

La tactique spécifique repose sur les quelques notions simples suivantes. On représente une situation de jeu par la liste des effectifs des tas, écrits en numération binaire. Par exemple, la configuration initiale habituelle est notée $(111, 101, 11, 1)$, soit l'équivalent binaire de $(7, 5, 3, 1)$. Une situation est *paire* si, pour tout n, le nombre de bits de rang n égaux à 1 est pair. C'est le cas pour la situation ci-dessus, qui comporte deux bits de rang 2 égaux à 1, deux bits de rang 1 égaux à 1 et quatre bits de rang zéro égaux à 1; ceci se remarque immédiatement si les effectifs sont disposés en une colonne:

$$\begin{array}{cc} 7 & 111 \\ 5 & 101 \\ 3 & 11 \\ 1 & 1 \end{array}$$

[10] Si on entre en contact avec un objet très chaud, le recul réflexe permettra d'éviter une brûlure; par contre, un recul conscient et motivé interviendrait trop tard.

Chaque colonne de chiffres comporte un nombre pair de 1.

Une situation qui n'est pas paire est dite *impaire*. On démontre facilement deux propriétés intéressantes:

- Tout coup licite joué au départ d'une situation paire conduit à une situation impaire.
- Au départ d'une situation impaire, il existe au moins un coup licite conduisant à une situation paire.

Il est donc possible au joueur confronté à une situation impaire d'adopter une stratégie telle que, jusqu'à la fin de la partie, son adversaire soit toujours confronté à une situation paire, dont la dernière sera inévitablement la situation vide, perdante.

Il est donc facile, pour un humain comme pour un programme, de jouer à Nim de manière optimale, même si le nombre d'allumettes au départ est élevé; c'est seulement si le joueur ne connaît pas la tactique qu'il recourra à une méthode d'intelligence artificielle, telle la recherche systématique dans un arbre modélisant le jeu. L'intelligence artificielle constitue donc le second choix et n'est adoptée que pour les problèmes où une tactique *ad hoc* n'est pas connue.

10.2.5. *Echecs et Hexapion*

Le célèbre jeu d'Echecs est intéressant parce qu'aucune tactique complète n'est connue, même si des centaines de stratégies de jeu, plus ou moins partielles, ont été proposées; les joueurs sont donc obligés de réfléchir. En début de partie, les joueurs utilisent beaucoup leur culture; il existe des "bibliothèques" d'ouvertures dont plusieurs siècles d'expérience ont prouvé l'efficacité. En fin de partie, il est parfois possible d'utiliser une stratégie algorithmique permettant, par exemple, à un Roi et une Tour de vaincre à coup sûr un Roi isolé. C'est sans doute en milieu de partie que le joueur aura le plus tendance à considérer, explicitement ou non, un arbre de jeu, nécessairement très partiel; en fait, le joueur examine les conséquences, à plus ou moins court terme, de quelques coups "prometteurs". Les Echecs constituent en théorie un bon exemple d'investigation des méthodes de l'intelligence artificielle mais la complexité même du jeu le rend peu approprié à cette brève introduction.

La difficulté quantitative disparaît si l'on considère une version simplifiée à l'extrême du jeu d'Echecs, à savoir l'Hexapion. Ce jeu se pratique sur un "échiquier" de trois cases sur trois. Chaque joueur dispose initialement de trois pions, posés sur la rangée la plus proche de lui. Comme aux Echecs, le pion avance en ligne droite (d'une case) et prend en diagonale. Pour éviter le risque de nullité, on convient qu'un joueur peut gagner de deux manières différentes:

- En menant un pion "à dame", c'est-à-dire sur la rangée la plus proche de l'adversaire.

- En mettant l'adversaire dans l'impossibilité de jouer.

Quatre parties d'Hexapion (pas nécessairement optimales) sont représentées à la figure 10.6. Notons que, comme aux Echecs, ce sont toujours les Blancs qui jouent les premiers.

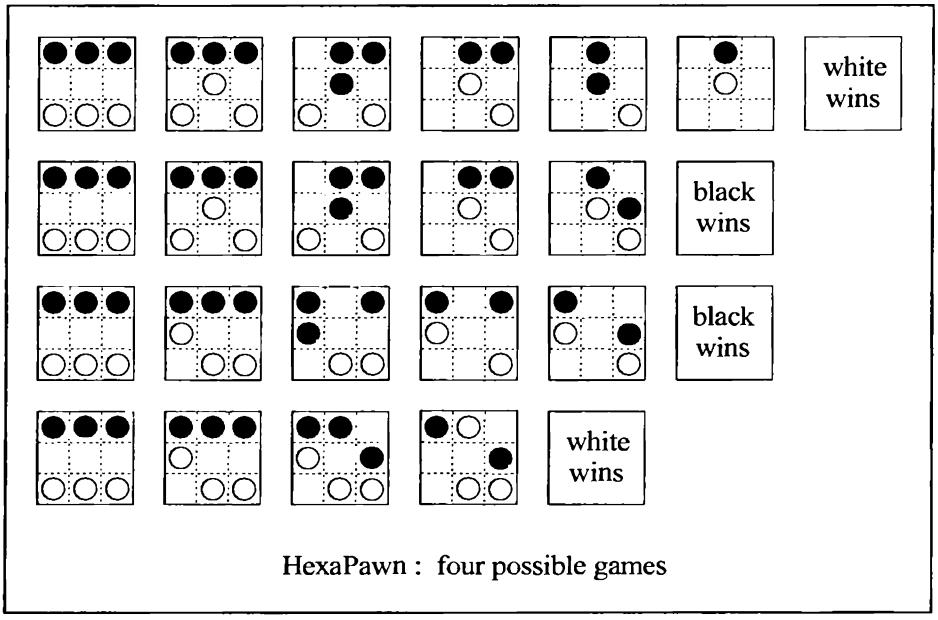


Figure 10.6 *Quatre parties d'Hexapion*

Il y a seulement quelques centaines de parties possibles à ce jeu; on pourrait donc concevoir un programme jouant de manière optimale par simple exploration d'une base de données comportant toutes les parties possibles. Une approche plus intéressante consiste à structurer cette base de données en un arbre de jeu. La figure 10.7 représente un fragment important de cet arbre; chaque nœud représente une configuration possible et chaque arc représente un coup possible.

L'arbre partiel de la figure 10.7 permet déjà d'apprendre beaucoup à propos d'Hexapion, et notamment qu'il existe une stratégie gagnante pour les Noirs.[11] Au départ, les Blancs ont deux possibilités, dont nous allons explorer les conséquences: avancer un pion latéral, ou avancer le pion central.

[11] En fait, la plupart des nœuds omis dans cet arbre correspondent à des configurations symétriques de celles associées à des nœuds présents dans l'arbre, ou encore à des nœuds frères d'une configuration gagnante, et donc inutile si on suppose qu'un joueur détecte systématiquement une victoire atteignable en un seul coup.

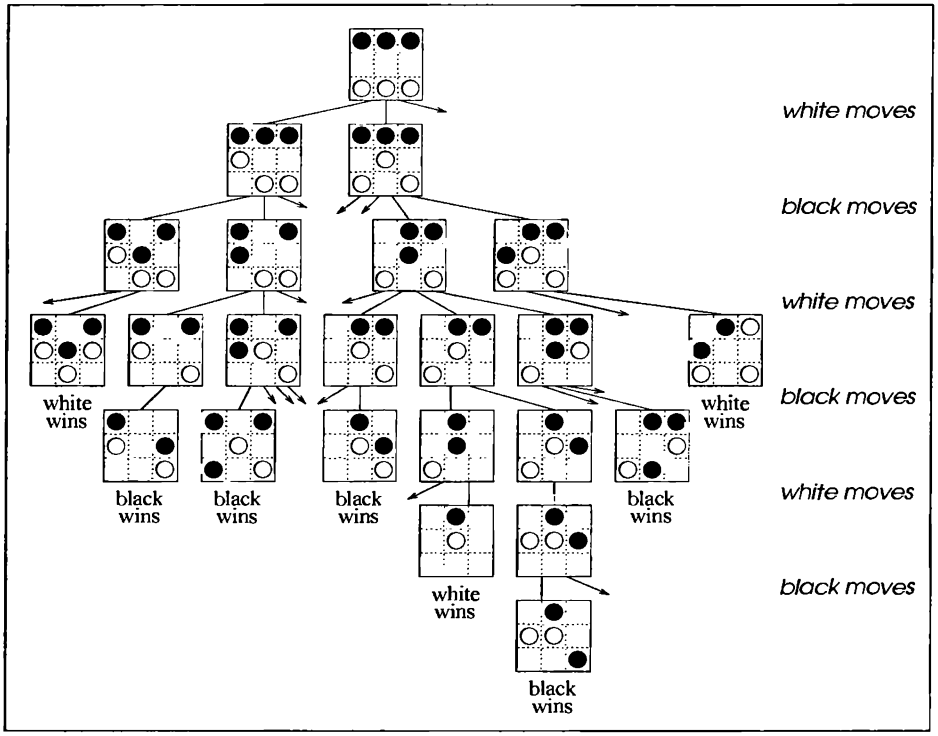


Figure 10.7 *Arbre partiel de jeu pour Hexapion*

- Supposons que les Blancs avancent le pion de gauche. Si les Noirs prennent immédiatement, il reste aux Blancs deux possibilités, dont aucune ne permet d'empêcher à coup sûr la victoire des Noirs :
  - Les Blancs prennent le pion avancé des Noirs, mais ces derniers peuvent gagner immédiatement en bloquant les Blancs.
  - Les Blancs avancent un de leurs pions (deux possibilités existent mais une seule est illustrée). Les Noirs répliquent en conduisant leur pion avancé à dame.
- Si les Blancs avancent le pion central, les Noirs ont à nouveau intérêt à prendre immédiatement. Après cela, les Blancs ont deux possibilités, dont aucune ne leur assure le gain :
  - Les Blancs prennent le pion avancé (deux possibilités), mais les Noirs peuvent provoquer un blocage (si une colonne est vide) ou avancer

le pion latéral qui leur reste, et le mener à dame au coup suivant (voir figure).

- Les Blancs avancent un de leurs pions, mais les Noirs peuvent alors immédiatement mener à dame l'un des leurs.

Cette analyse a mis en évidence une stratégie gagnante pour les Noirs : tous les mouvements possibles sont pris en compte pour les Blancs, seul le meilleur est considéré pour les Noirs. On pourrait de même étudier les stratégies possibles pour les Blancs, mais ceux-ci ne peuvent gagner que si les Noirs commettent une erreur.

Ces considérations stratégiques ne nous intéressent pas pour elles-mêmes; le point de vue de l'intelligence artificielle est qu'une stratégie gagnante peut non seulement être mise en œuvre par un programme, mais aussi, de manière plus ou moins explicite, inférée par un programme. Nous montrerons ceci dans la suite de ce chapitre; nous verrons aussi comment un programme peut apprendre au contact de son adversaire et devenir optimal après avoir perdu un certain nombre de parties.

10.3. Pour en savoir plus

Nous avons étudié brièvement dans ce chapitre introductif quelques aspects de l'intelligence artificielle et observé que ce concept était difficile à cerner. Dans les chapitres suivants nous abordons des points plus concrets, dans lesquels la logique joue un rôle important. Nous introduisons la programmation logique, qui permet de modéliser un univers particulier en une base de connaissances, et de déterminer ce qui peut être déduit de cette base de connaissances. La programmation logique permet de résoudre les problèmes les plus variés; nous l'utilisons notamment pour résoudre des énigmes amusantes mais non évidentes, et aussi pour écrire un programme simulant l'apprentissage du jeu d'Hexapion. Dans un autre registre, nous abordons le problème de la déduction automatique, que nous illustrons en vérifiant des propriétés de programmes. L'intelligence artificielle comporte beaucoup d'autres domaines, comme les systèmes experts, la planification, l'analyse et la génération de discours en langage naturel, l'apprentissage par réseaux neuronaux ou par arbres de décision, la vision artificielle, etc. De nombreux ouvrages ont été consacrés à l'intelligence artificielle; citons notamment [Dean et al. 95, Ginsberg 93, Luger and Stubblefield 98, Winston 92]. Mentionnons en français quelques classiques [Farreny et Ghallab 87, Haton et al. 91, Laurière 87]; de nouvreux ouvrages paraissent régulièrement, entre autres dans la présente collection "Langue, Raisonnement, Calcul".

Chapitre 11
Principes de la programmation logique

Nous avons observé que le raisonnement implique habituellement la définition d'un espace structuré d'objets et l'exploration de cet espace; les exemples d'Hexapion, de l'énigme des Trois Etudiants et du problème des Tours de Hanoï sont typiques à cet égard. Il est dès lors nécessaire de pouvoir représenter des objets et déterminer l'espace qu'ils forment, qui est l'ensemble de tous les objets; il faut également pouvoir spécifier la structure de cet espace et les techniques permettant de l'explorer. La logique formelle est le langage généralement le mieux adapté à la représentation de la connaissance et la programmation logique est un moyen de manipuler cette connaissance. Le système PROLOG est une réalisation imparfaite des techniques de la programmation logique, mais il apporte une solution simple et économique à la plupart des problèmes de représentation et de manipulation de la connaissance. A ce titre, il est un outil privilégié de l'intelligence artificielle.

11.1. Concepts de base

11.1.1. *Les termes*

Le premier concept en PROLOG et en intelligence artificielle est celui d'objet. N'importe quoi est un objet : nombres, mots, personnes, ensembles, configurations de jeu d'échecs, etc. Cependant, le calcul et le raisonnement (automatisé) ne manipulent pas directement des objets mais leurs représentations. Pour des objets "abstraits" tels les nombres, la distinction ne semble pas très importante et il est fréquent de confondre, par exemple, le nombre 13 et sa représentation sous forme des deux caractères 1 et 3. On sait cependant que certaines

représentations (numérations binaire et décimale) sont plus commodes que d'autres (écriture en toutes lettres, chiffres romains). Pour des entités plus "concrètes", comme les personnes par exemple, la différence entre l'objet et sa représentation est plus importante même si elle n'est pas toujours explicitement soulignée. Les personnes peuvent être désignées par des mots (noms et prénoms) ou des nombres (numéros d'ordre, matricules), etc.

En PROLOG comme dans beaucoup de langages de programmation, les objets sont représentés par des *termes*. Les *termes de base* ou *termes atomiques* ou *atomes* sont de deux sortes; un objet est désigné par une *constante* si la liaison entre l'objet et sa représentation présente un certain caractère de permanence; sinon, il est désigné par une *variable*. Plus précisément, PROLOG *identifie* les constantes aux objets qu'elles représentent; en fait, PROLOG ignore les objets et ne connaît que les représentations. Les variables, par contre, sont liées à des objets, ou plutôt aux constantes qui représentent ces objets, mais cette liaison peut évoluer en cours d'exécution d'un programme. Une même variable (un "mot" commençant par une majuscule ou par le caractère spécial _) peut selon les moments et les occurrences, représenter des objets divers; de même, un objet unique peut être lié à des variables distinctes.[1] Les constantes de PROLOG sont essentiellement les nombres, tels `0`, `-3` et `2.567`, et les symboles, comme `abc`, `+` et `rs232`.

Un *foncteur* est un symbole que l'on utilise pour construire des *termes composés*. Chaque foncteur a une *arité* qui est un nombre naturel. Etant donné un foncteur `f` d'arité n et n termes $\tau_1, \ldots, \tau_n$ (atomiques ou composés), l'expression `f(`τ_1`, ... ,`τ_n`)` est un terme composé. Les expressions `m(a,b,c)`, `m(f(a,b),a(c,g(2,d)))` et `+(3,*(4,5))` sont des exemples de termes composés *fondamentaux*, c'est-à-dire ne comportant pas de variable. Comme les constantes,[2] les termes fondamentaux sont, en PROLOG, identifiés aux objets qu'ils représentent. Du point de vue de PROLOG, les termes fondamentaux *sont* les objets. C'est sous sa seule responsabilité que l'utilisateur PROLOG associe à un terme fondamental un autre objet; il est assez naturel d'associer aux termes `-3` et `+(3,*(4,5))` le nombre -3 et l'expression arithmétique $3 + (4 * 5)$ mais ces associations sémantiques ne sont pas obligatoires. Les constantes lexicales (non numériques) ne représentent qu'elles-mêmes. Les termes composés sont des arbres dont les feuilles sont des atomes et les nœuds internes des foncteurs. Par exemple, `m(a,b,c)` est un arbre ternaire dont la racine est (étiquetée par) `m` et dont les feuilles sont (étiquetées par), de gauche à droite, `a`, `b` et `c`. A une même constante symbolique `f` peuvent correspondre plusieurs foncteurs, que l'on distingue par leurs arités respectives.

Supposons que dans une base de données, une personne soit représentée par son nom, son prénom et son âge. Une représentation typique en PROLOG

[1] La notion de variable, ici assez floue, sera approfondie plus loin.

[2] Les constantes sont en fait les termes fondamentaux atomiques.

serait, par exemple, `p(gates,bill,43)`. Ici, `p` pourrait rappeler à l'utilisateur qu'un terme de ce type est associé à une personne, mais tout autre symbole pourrait convenir. Dans la vie courante comme en mathématique et dans d'autres sciences, une structure générale et importante est celle de *liste*. On pourrait par exemple représenter des personnes par des "structures anonymes" telles que `[gates,bill,43]`. PROLOG permet cette notation commode mais l'interprète comme une variante syntaxique d'un terme composé classique, à savoir `.(gates,.(bill,.(43,[])))`. La constante symbolique `[]` désigne la liste vide. La constante symbolique "`.`" est un foncteur binaire (d'arité 2) employé pour apparier deux termes. Une liste non vide est, par définition, une paire dont la *tête* (premier composant) représente le premier élément de la liste et dont le *reste* (second composant) est la liste des éléments restants. En PROLOG, la paire `.(a,b)` peut aussi s'écrire `[a|b]`. On en déduit que, par exemple, le terme `[gates|[bill|[]]]` est aussi une variante syntaxique du terme `.(gates,.(bill,[]))`, ou encore du terme `[gates,bill]`. Soulignons que, les termes composés étant des arbres, les foncteurs (et en particulier le foncteur "`.`") ne sont pas associatifs. Les termes `.(gates,.(bill,[]))` et `.(.(gates,bill),[])` sont donc bien distincts, de même que les termes `f(a,f(b,c))` et `f(f(a,b),c)`.

L'ensemble des termes fondamentaux, c'est-à-dire, pour PROLOG, l'ensemble de tous les objets, semble très rudimentaire. Le concept de terme est néanmoins suffisamment flexible pour permettre de représenter commodément n'importe quel objet de la vie courante ou d'un domaine particulier de la connaissance. Par exemple, une configuration du jeu d'Hexapion peut être représentée par le terme :

```
hp(c(1,1,v),c(1,2,n),c(1,3,n),
   c(2,1,n),c(2,2,b),c(2,3,v),
   c(3,1,b),c(3,2,v),c(3,3,b))
```

dans lequel chaque c-triplet représente une case du tableau de jeu; les deux premières composantes identifient la position de la case (ligne et colonne) et la troisième identifie le contenu (`b` pour pion blanc, `n` pour pion noir, `v` pour case vide). D'autres représentations sont possibles et tout aussi commodes.

Chaque terme fondamental est un objet précis. On peut voir un terme non fondamental comme un objet générique, élément d'un certain ensemble. Cet ensemble sous-jacent est l'ensemble de tous les objets dans le cas d'une variable, mais devient un ensemble plus restreint dans le cas d'un terme composé. Les éléments de cet ensemble sont appelés les *instances* du terme. Tout objet (tout terme fondamental) est une instance de n'importe quelle variable. Les instances du terme `m(f(a,X),f(X,b),Y)` s'obtiennent en remplaçant `X` et `Y` par des termes fondamentaux quelconques. Des exemples d'instances de `m(f(a,X),f(X,b),Y)` sont `m(f(a,a),f(a,b),c)` (on a intancié `X` en `a` et `Y` en `c`) et `m(f(a,g(c)),f(g(c),b),g(c))` (on a intancié `X` et `Y` en `g(c)`); un contre-exemple est `m(f(a,b),f(c,b),d)` (les deux occurrences de `X` ont été

remplacées par deux termes différents).
Remarque. Un foncteur est toujours une constante symbolique; une expression telle que `X(a,b)` n'est donc pas un terme.

On peut résumer comme suit l'association que PROLOG fait entre un terme et l'objet qu'il représente:

- Les objets de PROLOG sont les termes fondamentaux.
- A tout terme est associé un ensemble d'instances, qui est un ensemble de termes fondamentaux.
- Un terme représente une instance générique, ou une instance particulière, ou l'ensemble des instances associé à ce terme.
- L'ensemble des instances d'un terme fondamental se réduit à ce terme; en conséquence, un terme fondamental représente un seul objet, lui-même.
- L'ensemble des instances d'une variable est l'ensemble de tous les termes fondamentaux.

11.1.2. *Faits et règles*

Un programme PROLOG est une collection de *clauses*; une clause est un *fait* ou une *règle*. Syntaxiquement, un fait est simplement un terme suivi d'un point. Sémantiquement, un fait fondamental affirme l'appartenance d'un n-uplet à une relation. Par exemple, le fait

```
r(a,b,c).
```

affirme que le n-uple `(a,b,c)` appartient à la relation ternaire `r`. (Savoir ce que représentent la relation `r` et les éléments `a`, `b` et `c` est une autre question, qui ne concerne pas la sémantique de PROLOG.) Un identificateur utilisé comme foncteur d'un fait est un *prédicat*. Comme en logique, un prédicat représente une relation. Les phrases "Le triplet `(a,b,c)` appartient à la relation (représentée par) `r`" et "`r(a,b,c)` est vrai" sont donc équivalentes. L'*arité* du prédicat est son nombre d'argument(s). Deux prédicats d'arités différentes sont différents même s'ils portent le même nom. Les prédicats d'arité 0 sont les *propositions*; on les écrit naturellement sans parenthèses. Par exemple,

```
z.
```

est un fait, affirmant que la proposition `z` est vraie. Tout fait fondamental affirme qu'un prédicat est vrai.[3]

Il est possible de décrire des relations finies en utilisant exclusivement des faits fondamentaux mais cela ne suffit pas pour décrire des relations infinies ni, en pratique, des relations finies de grande taille. Nous le montrons simplement en introduisant une structure de donnée essentielle en intelligence artificielle, le

[3]Au sens strict, `r` est un symbole prédicatif et `r(a,b,c)` est une forme prédicative. Dans l'usage courant, les deux expressions peuvent être qualifiées de prédicats.

graphe, c'est-à-dire la relation binaire. Nous avons vu qu'une relation binaire, et donc un graphe, se représentait par un prédicat binaire. Si on souhaite manipuler simultanément plusieurs graphes — c'est fréquent en intelligence artificielle — on prévoira un argument supplémentaire qui identifiera le graphe. En outre, on souhaite parfois introduire dans le domaine de la relation, c'est-à-dire dans l'ensemble des nœuds du graphe, des éléments qui n'interviennent dans aucun couple de la relation, c'est-à-dire dans aucun arc du graphe.[4] Un graphe (orienté) G dont les nœuds sont $n_1, \ldots, n_6$ et dont les arcs sont (n_1, n_3), (n_1, n_5), (n_2, n_3), (n_2, n_4), (n_4, n_5), (n_4, n_6) et (n_5, n_6) est naturellement décrit par un ensemble de faits fondamentaux :

```
node(g,n1).          arc(g,n1,n3).    arc(g,n1,n5).
node(g,n2).          arc(g,n2,n3).    arc(g,n2,n4).
node(g,n3).
node(g,n4).          arc(g,n4,n5).    arc(g,n4,n6).
node(g,n5).          arc(g,n5,n6).
node(g,n6).
```

Le premier argument g n'est nécessaire que si d'autres graphes sont aussi modélisés; rappelons que le nom du graphe doit commencer par une minuscule; tout identificateur commençant par une majuscule est, conventionnellement, une variable. Un *chemin* dans un graphe est une suite de nœuds telle que deux éléments consécutifs sont respectivement l'origine et l'extrémité d'un arc du graphe. Le premier élément d'un chemin est l'origine du chemin. Le dernier élément d'un chemin fini est l'extrémité du chemin. Un *cycle* est un chemin fini dont l'origine et l'extrémité coïncident. Un graphe *acyclique* est un graphe sans cycle. Un graphe acyclique fini, tel celui représenté ci-avant, comporte un nombre fini de chemins. Si nous nous intéressons à cet ensemble, il est possible de le décrire par des faits fondamentaux. L'un de ces faits serait :

```
path(g,[n1,n5,n6]).
```

Il est néanmoins fastidieux et inutile de construire cette description de manière explicite, car elle se déduit aisément de la description du graphe lui-même. L'une des fonctions de base du système PROLOG est de dispenser son utilisateur de la construction explicite de telles descriptions; une description implicite, très simple, suffit. Elle est basée sur deux constatations. Tout d'abord, si `X` est un nœud, alors `[X]` est un chemin ("clause de base"). Ensuite, si `(X,Y)` est un arc et si `[Y|Ys]` est un chemin, alors `[X,Y|Ys]` est un chemin ("clause

[4] Il est question ici de graphe orienté. Chaque arc comporte une origine et une extrémité, dont les rôles ne sont pas interchangeables. Dans le cas d'un graphe non orienté, un arc (ou une arête) est une paire de nœuds plutôt qu'un couple de nœuds. On peut toujours assimiler un graphe non orienté à un graphe orienté symétrique : l'arête $\{a, b\}$ est alors modélisée par les deux couples (a, b) et (b, a). Dans la suite, il sera surtout question de graphes orientés.

inductive, ou récursive").[5] Ces deux constatations affirment qu'une certaine liste de nœuds est un chemin à certaines conditions. En PROLOG, on formalise les affirmations conditionnelles par des *règles*. La notion de chemin se traduit par les deux règles suivantes :[6]

```
path(G,[X]) :- node(G,X).
path(G,[X,Y|Ys]) :- node(G,X), arc(G,X,Y), path(G,[Y|Ys]).
```

La signification logique d'une clause PROLOG, c'est-à-dire sa sémantique déclarative, est très simple. Une règle telle que:

```
A :- B,C,D.
```

exprime que, si `B`, `C` et `D` sont vrais, alors `A` est vrai.[7] Le prédicat `A` est la *tête* de la clause, la suite de prédicats `B,C,D` en est le *corps*. Les variables sont universellement quantifiées. Par exemple, la première règle du programme ci-avant affirme que pour tout graphe `G` et pour tout nœud `X` de ce graphe, la liste `[X]` est un chemin de `G`. Un fait est simplement une clause dont le corps est vide. On écrit "`A.`" au lieu de "`A :- .`".

11.1.3. *Questions*

Un programme PROLOG peut être vu comme une base de données relationnelle (descriptions explicites de relations) enrichie de règles (descriptions implicites de relations). On peut appeler cela une *base de connaissances*. Les bases de données et de connaissances peuvent être interrogées. Voici par exemple quelques questions envisageables dans le cas du programme relatif au graphe g, enrichi de la notion implicite de chemin.

- Quels sont les nœuds du graphe g ?
- Est-ce que `[n1,n5,n6]` est un chemin de g ?
- Quels sont les chemins de g dont l'origine est `n1` ou `n2` et dont l'extrémité est `n6` ?

La réponse à la première question est immédiate puisque les nœuds sont explicitement mentionnés (sous forme de faits fondamentaux) dans le programme. La seconde question appelle une réponse par oui ou non. Il est clair que l'usage

---

[5] On pourrait ajouter une "clause de fermeture", à savoir, tout chemin s'obtient par application des clauses de base et récursive. Ceci est implicite en PROLOG.

[6] Notons ici une convention habituelle mais non impérative : quand une variable désigne une liste, on peut attirer l'attention sur ce point en utilisant un identificateur se terminant par "s". Observons aussi l'usage (impératif) de majuscules : notre description implicite de la notion de chemin dans un graphe est valable pour tout graphe et pour tout chemin.

[7] En logique, une formule de ce type est notée $(B \wedge C \wedge D) \Rightarrow A$ ou $(A \vee \neg B \vee \neg C \vee \neg D)$; on appelle "clause" une formule de ce type, d'où le choix naturel du mot "clause" pour désigner un élément d'une description en PROLOG, c'est-à-dire une instruction d'un programme PROLOG.

de la seconde règle relative aux chemins est indispensable pour répondre à la question. Un cas particulier de cette règle, ou plutôt une *instance* de cette règle est :

```
path(g,[n1,n5|[n6]]) :-
  node(g,n1), arc(g,n1,n5), path(g,[n5|[n6]]).
```

ou encore, de manière plus lisible :

```
path(g,[n1,n5,n6]) :-
  node(g,n1), arc(g,n1,n5), path(g,[n5,n6]).
```

La question principale se réduit à la suite des trois *sous-questions* `node(g,n1)`, `arc(g,n1,n5)` and `path(g,[n5,n6])`. Nous observons aussi que :

```
node(g,n1).
arc(g,n1,n5).
```

sont des faits fondamentaux présents dans la base de connaissance, donc la question principale est maintenant réduite à la sous-question : `path(g,[n5,n6])`. On utilise à présent une autre instance de la règle récursive relative aux chemins, à savoir :

```
path(g,[n5,n6]) :- node(g,n5), arc(g,n5,n6), path(g,[n6]).
```

A nouveau, on répond affirmativement aux deux premières nouvelles sous-questions `node(g,n5)` et `arc(g,n5,n6)`, qui correspondent à des faits fondamentaux du programme; seule la troisième sous-question :

```
path(g,[n6]).
```

subsiste. On utilise maintenant une instance de la règle de base relative aux chemins, à savoir :

```
path(g,[n6]) :- node(g,n6).
```

Cela conduit à la sous-question :

```
node(g,n6).
```

Cette sous-question correspond à un fait du programme, donc la réponse à cette sous-question, ainsi qu'aux sous-questions antérieures et à la question originale, est `oui`.

La troisième question est un peu moins simple. On peut construire tous les chemins issus de `n1` ou `n2` et ensuite éliminer ceux dont l'extrémité n'est pas `n6`; on peut aussi construire tous les chemins d'extrémité `n6` et ensuite éliminer ceux dont l'origine n'est ni `n1` ni `n2`. Il n'est pas évident a priori de choisir entre ces deux stratégies. On observe aussi un certain "gaspillage" : on construit une collection relativement importante de chemins pour n'en garder en fin de compte qu'une partie. Il convient de souligner que ce phénomène est omniprésent en intelligence artificielle ... et même en intelligence naturelle. Si la résolution d'un problème implique une démarche intelligente, elle implique aussi le risque de se tromper, ce qui, dans notre cadre, correspond à la construction de "solutions" non satisfaisantes, qui seront finalement rejetées.

Les quelques exemples envisagés au chapitre précédent, quoique simples, mettent bien en évidence le phénomène. Des jeux tels que Nim et Hexapion comportent des risques d'erreurs; un joueur, même raisonnablement compétent, est susceptible d'envisager plusieurs coups malheureux avant de penser à un coup favorable. Dans le cas de l'énigme des Trois Etudiants, on peut envisager diverses associations Etudiant-Nationalité-Sport avant de découvrir la bonne. En fait, l'intelligence implique la faculté de reconnaître les erreurs rapidement. Les bons joueurs d'Echecs envisagent relativement peu de coups avant d'en choisir un, mais même un champion ne pourra gagner face à un joueur relativement compétent s'il joue systématiquement le premier coup qui lui vient à l'esprit. Il n'empêche que l'aptitude à éliminer rapidement les "solutions" non satisfaisantes est très précieuse. En PROLOG, on veillera à construire le moins possible de solutions insatisfaisantes; on essaiera aussi d'éliminer une solution partielle peu prometteuse avant d'avoir achevé sa construction. D'une manière générale, concevoir un espace de solutions potentielles et l'explorer efficacement est un des buts majeurs de l'intelligence artificielle.

11.2. Sémantique

La programmation logique met "la logique en action"; elle apparie la programmation et la logique. Elle comporte donc une sémantique opérationnelle (comment le système PROLOG explore-t-il et exploite-t-il une base de connaissance, un programme logique?) et une sémantique déclarative (quels faits sont tenus et reconnus pour vrais par le système PROLOG?). Les deux types de sémantiques sont introduits ici.

11.2.1. *Sémantique opérationnelle*

Comme l'ont déjà suggéré les exemples élémentaires de traitement de graphe, le système PROLOG répond aux questions qui lui sont posées en construisant et en explorant une structure arborescente. Une ramification se produit quand plusieurs clauses sont susceptibles d'être utilisées pour la réduction d'une sous-question donnée. Une branche se termine positivement quand une solution est trouvée, après réduction de toutes les sous-questions (branche-succès); une branche se termine négativement lorsqu'aucune clause ne permet de réduire une certaine sous-question (branche-échec). Il se peut aussi qu'une branche ne se termine jamais (branche infinie). La *sémantique opérationnelle* de PROLOG est la description précise de la procédure de construction et d'exploitation de cette structure arborescente que l'on nomme *arbre de recherche*. Cette notion a été étudiée sur le plan conceptuel dans le volume précédent (cf. § II.5.6); rappelons qu'elle repose sur deux algorithmes fondamentaux de la logique, l'algorithme de résolution et l'algorithme d'unification, étudiés dans le premier volume (cf. §§ I.4.4, I.6.4 et I.8.3). Nous illustrons ici, de manière plus concrète,

la notion essentielle d'arbre de recherche, en abordant un exemple de traitement de listes; ce domaine est spécialement intéressant dans ce contexte parce que les listes constituent probablement la structure de donnée la plus utilisée en programmation symbolique et en intelligence artificielle.

Les mécanismes de base pour la construction de listes sont très simples. D'une part, on dispose d'une constante spéciale, notée `[]`, correspondant à la liste vide; d'autre part, étant donné un objet quelconque `X` et une liste quelconque `Xs`, le terme `.(X,Xs)`, plus communément noté `[X|Xs]`, désigne la liste dont le premier élément est `X` et dont les autres éléments forment la liste `Xs`. Par exemple, si `X = a` et `Xs = [1,2,3]`, alors `[X|Xs] = [a,1,2,3]`. Une construction plus générale est la *concaténation*. La concaténation de `[a,b,c]` et `[1,2]` est `[a,b,c,1,2]`. La concaténation est classiquement définie de manière récursive. D'abord, la concaténation de la liste vide et d'une liste quelconque `Bs` est simplement `Bs`. Ensuite, si la concaténation de `As` et `Bs` (listes quelconques) est `Cs`, alors, pour tout objet `A`, la concaténation de `[A|As]` et `Bs` est `[A|Cs]`. Ces deux postulats suffisent à définir la concaténation. Le programme PROLOG correspondant permet de définir un prédicat ternaire, classiquement nommé `append`; ce programme est donné à la figure 11.1. Sur base du prédicat `append`, on peut définir un nouveau prédicat binaire `prefix` au moyen d'un simple fait (non fondamental) interprété comme suit: `Xs` est un préfixe de `Ys` si une liste `Zs` existe telle que la concaténation de `Xs` et `Zs` est `Ys`. Enfin, la figure 11.1 montre aussi l'arbre de recherche engendré par PROLOG quand la question `prefix(Xs,[a,b])` lui est soumise.

L'arbre de recherche est construit selon des règles élémentaires mais strictes, qu'il convient d'analyser en détail. Observons d'abord que les nœuds et les arcs de l'arbre sont tous munis d'une étiquette. L'étiquette de la racine est la question soumise au système. Par rapport à cette question, le programme comporte une seule clause *unifiante*, c'est-à-dire une clause dont la tête a pour prédicat celui de la question.[8] En conséquence, la racine aura un seul fils. L'arc unissant la racine à ce fils est étiquetée par l'*unificateur principal* permettant d'égaliser le (premier) prédicat étiquetant la racine, soit `prefix(Xs,[a,b])` et la tête de la clause unifiante, soit `prefix(Xs,Ys)`.[9] Rappelons que les variables

[8]Une question ou une sous-question peut comporter plusieurs prédicats; c'est alors pour le premier de ceux-ci que l'on cherche des clauses unifiantes.

[9]La notion d'unificateur principal a été définie au paragraphe I.6.4.8; le paragraphe I.6.4.10 présente l'algorithme d'unification, qui détermine si deux termes admettent des unificateurs principaux et, si oui, fournit l'un d'eux. En première lecture, il est suffisant de se souvenir de ceci. Une *substitution* sur un ensemble V de variables est une fonction σ qui à toute variable de V associe un terme. Le domaine de la fonction σ s'étend immédiatement à l'ensemble $\mathcal{T}_V$ des termes dont toutes les variables sont dans V; l'image $\sigma(t)$ d'un terme t par σ s'obtient en remplaçant toute variable v de t par le terme $\sigma(v)$. Etant donnés deux termes $t_1, t_2 \in \mathcal{T}_V$ et une substitution σ de domaine V, on dit que σ est un *unificateur* de t_1 et t_2 si $\sigma(t_1) = \sigma(t_2)$. Un unificateur σ de t_1 et t_2 est dit *principal* si pour tout unificateur ρ de t_1 et t_2 il existe une substitution ϕ telle que $\phi \circ \sigma = \rho$.

```
prefix(Xs,Ys) :- append(Xs,Zs,Ys).
append([],Bs,Bs).
append([A|As],Bs,[A|Cs]) :- append(As,Bs,Cs).
```

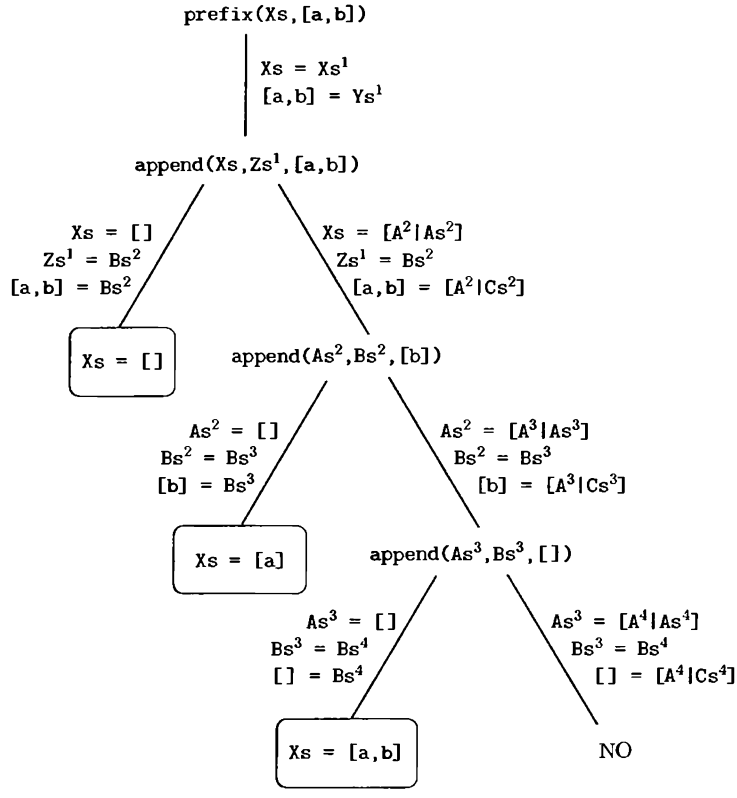


Figure 11.1 *Programme et arbre de recherche*

des clauses sont quantifiées universellement; l'identité de nom de la variable `Xs` de la question et de la variable `Xs` de la tête de la clause est donc accidentelle et nous rebaptisons la seconde variable en $\texttt{Xs}^1$. Plus généralement, les variables des clauses sont affectées d'un exposant correspondant au niveau de l'arbre où elles sont utilisées. L'unificateur principal adéquat doit ici égaliser la question `prefix(Xs,[a,b])` et la tête de la clause unifiante $\texttt{prefix(Xs}^1\texttt{,Ys}^1\texttt{)}$; il se composera donc des deux équations $\texttt{Xs} = \texttt{Xs}^1$ et $\texttt{[a,b]} = \texttt{Ys}^1$. En fait, la clause unifiante signifie que, pour obtenir $\texttt{prefix(Xs}^1\texttt{,Ys}^1\texttt{)}$, il est suffisant d'obtenir $\texttt{append(Xs}^1\texttt{,Zs}^1\texttt{,Ys}^1\texttt{)}$. En tenant compte de l'unificateur, cette

clause permet la réduction de la question `prefix(Xs,[a,b])` à la sous-question `append(Xs,Zs`^{1}`,[a,b])`; cette sous-question sera l'étiquette du nœud-fils de la racine de l'arbre.[10]

La deuxième étape de la construction de l'arbre de recherche est la réduction de la sous-question `append(Xs,Zs`^{1}`,[a,b])`. Il existe deux clauses unifiantes (dont la tête comporte le prédicat append), donc le nœud étiqueté par la sous-question à réduire aura deux fils. Le corps de la première clause unifiante (branche de gauche) est vide, ce qui signale que cette branche sera une branche succès. L'unificateur adéquat comporte ici les trois équations car le prédicat append est d'arité 3; on a `Xs` = `[]`, `Zs`^{1} = `Bs`^{2} et `[a,b]` = `Bs`^{2}. Cet unificateur égalise la sous-question `append(Xs,Zs`^{1}`,[a,b])` et la tête de la clause unifiante `append(Xs,Bs`^{2}`,Bs`^{2}`)`. La solution correspondant à cette branche-succès est obtenue au moyen des unificateurs étiquetant cette branche; on obtient ainsi `Xs` = `[]`, ce qui est correct: la liste vide est bien un préfixe de la liste `[a,b]`.

La troisième étape consiste à construire le second fils de la sous-question `append(Xs,Zs`^{1}`,[a,b])`. La tête de la clause unifiante s'écrit maintenant `append([A`^{2}`|As`^{2}`],Bs`^{2}`,[A`^{2}`|Cs`^{2}`])`; l'unificateur comporte les trois équations `Xs` = `[A`^{2}`|As`^{2}`]`, `Zs`^{1} = `Bs`^{2} et `[a,b]` = `[A`^{2}`|Cs`^{2}`]`. Le corps de la clause unifiante est `append(As`^{2}`,Bs`^{2}`,Cs`^{2}`)` qui, vu l'unificateur, se réduit à la sous-question `append(As`^{2}`,Bs`^{2}`,[b])`.[11] Ce prédicat étiquètera donc le second fils. Les étapes ultérieures construisent la descendance de ce dernier nœud, ce qui, comme l'indique la figure, produit deux solutions supplémentaires: `Xs = [a]` et `Xs = [a,b]`. On observe aussi que la dernière branche, c'est-à-dire la branche la plus à droite, est une branche-échec. La raison en est que la troisième équation du dernier unificateur est insoluble. En effet, l'égalité `[]` = `[A`^{4}`|Cs`^{4}`]` est impossible puisque le membre de gauche désigne une liste sans élément tandis que le membre de droite désigne une liste comportant au moins un élément.

Remarque. Nous avons mentionné que les variables d'une clause étaient, de manière implicite, quantifiées universellement. Précisons que la portée d'une variable est la clause qui la contient; on peut donc, par exemple, remplacer la clause:

```
prefix(Xs,Ys) :- append(Xs,Zs,Ys).
```

par la clause:

---

[10] Les variables apparaissant dans une question ne sont jamais renommées, c'est-à-dire affectées d'un exposant; en effet, ces variables ont été choisies par l'utilisateur du système, donc le système les emploiera pour communiquer les résultats du calcul à l'utilisateur. En PROLOG, les résultats d'un calcul sont les valeurs des variables (si elles existent) qui permettent de répondre affirmativement à la question de l'utilisateur. Les variables des clauses sont au contraire renommées à chaque utilisation au sein d'une même branche de l'arbre. On peut cependant avoir les mêmes noms dans des branches distinctes car deux branches distinctes n'interfèrent pas.

[11] De `[a,b]` = `[A`^{2}`|Cs`^{2}`]` on déduit `a` = `A`^{2} et `[b]` = `Cs`^{2}.

```
prefix(As,Bs) :- append(As,Cs,Bs).
```

qui est équivalente, mais pas par la clause :

```
prefix(As,Bs) :- append(Cs,Ds,Es).
```

qui est trop générale, ni par la clause :

```
prefix(As,Bs) :- append(As,Bs,Bs).
```

qui est trop restrictive. D'autre part, les variables d'une question sont quantifiées existentiellement; dans l'exemple développé plus haut, la question `prefix(Xs,[a,b])` s'interprète en "existe-t-il une liste `Xs` qui soit un préfixe de la liste `[a,b]` ?". PROLOG répond de manière affirmative, constructive et exhaustive à cette question, en proposant successivement pour `Xs` les valeurs `[]`, `[a]` et `[a,b]`.

Au premier abord, on constate que la construction de l'arbre de recherche est un processus fastidieux mais élémentaire. On pourrait décrire le noyau d'un système PROLOG en spécifiant de manière plus formelle la manière dont les étiquettes des nœuds et des arcs sont produites, ainsi que le mécanisme par lequel les solutions sont extraites et fournies à l'utilisateur. L'ordre dans lequel les manipulations sont faites est également important; une branche de gauche doit être terminée (par un succès ou par un échec) avant que la construction d'une branche plus à droite soit entamée. Cela a pour conséquence qu'une branche infinie provoque la non-terminaison du programme, et la non-production des résultats correspondant aux branches plus à droite éventuelles. Dans la suite du présent chapitre, nous illustrons par des exemples simples la *sémantique logique*, ou *déclarative*, de PROLOG, c'est-à-dire le lien logique existant entre les questions posées par un utilisateur et les réponses que PROLOG fournit au moyen du mécanisme de construction d'arbre que nous venons de décrire. D'autres exemples plus élaborés seront introduits au chapitre suivant; ils permettront de mieux illustrer l'intérêt de la programmation logique en intelligence artificielle.

11.2.2. *Sémantique déclarative*

La sémantique opérationnelle de PROLOG, esquissée au paragraphe précédent, permet de comprendre comment le système produit les réponses aux questions qui lui sont soumises, mais la portée de ces réponses, c'est-à-dire leur lien logique avec la question et le programme, se comprend mieux par le biais de la sémantique déclarative. En effet, d'un point de vue opérationnel, une clause telle que "`A :- B,C.`" traduit l'idée "pour obtenir `A`, on doit obtenir (au préalable) `B` et `C`". Le point de vue logico-déclaratif est "`A` est vrai si `B` et `C` sont vrais". La clause PROLOG est donc équivalente à la clause logique notée $(B \wedge C) \Rightarrow A$, ou encore $A \vee \neg B \vee \neg C$.

Du point de vue déclaratif, le programme décrit précédemment apparaît comme une théorie, c'est-à-dire un ensemble de postulats relatifs à une

structure, l'ensemble des listes en l'occurrence, munie d'opérations algébriques, ici la concaténation et le préfixe. Chaque branche-succès de l'arbre de recherche peut être vue comme une preuve d'un théorème correspondant à la solution associée à cette branche. Par exemple, la branche se terminant sur la solution `Xs = [a]` peut être lue, de bas en haut, comme une démonstration du fait que la liste `[a]` est un préfixe de la liste `[a,b]`. Nous reproduisons cette preuve sous une forme plus classique.

1. `append([],[Bs`^{3}`],[Bs`^{3}`])` ; (clause de base – append)
2. `append([],[b],[b])` ; (instantiation, ligne 1)
3. `append([A`^{2}`|As`^{2}`],Bs`^{2}`,[A`^{2}`|Cs`^{2}`])` si `append(As`^{2}`,Bs`^{2}`,Cs`^{2}`)` ; (clause inductive – append)
4. `append([a],[b],[a,b])` if `append([],[b],[b])` ; (instant., ligne 3)
5. `append([a],[b],[a,b])` ; (déduction, lignes 2 et 4)
6. `prefix(Xs`^{1}`,Ys`^{1}`)` if `append(Xs`^{1}`,Zs`^{1}`,Ys`^{1}`)` ; (clause – `prefix`)
7. `prefix([a],[a,b])` if `append([a],[b],[a,b])` ; (instant., ligne 6)
8. `prefix([a],[a,b])`. (déduction, lignes 5 et 7)

Pour l'arbre de la figure 11.1, on constate aisément que, d'une part, il possède trois branches succès et que, d'autre part, les étiquettes des trois feuilles correspondantes sont précisément les réponses à la question `prefix(Xs,[a,b])` étiquetant la racine de l'arbre. Il n'est pas évident que cette coïncidence entre l'approche opérationnelle et l'approche déclarative se produise aussi dans des cas plus complexes; cette question a donné lieu à toute une théorie dont nous donnons ici quelques éléments.

11.3. Approche théorique de la programmation logique

La programmation logique est un concept extrêmement ambitieux puisque l'idée sous-jacente consiste en un algorithme dont les données sont un ensemble de formules (le "programme" ou les "hypothèses") et une formule (la "question" ou la "conclusion"); cet algorithme détermine si la conclusion est conséquence logique des hypothèses et, si c'est le cas, construit une preuve de ce fait. Il y a bien certaines restrictions: les hypothèses sont des clauses de Horn (ou plutôt, des fermetures universelles de clauses de Horn) et la conclusion est un cube positif (ou plutôt, la fermeture existentielle d'un cube positif)[12] mais on conçoit, vu le théorème de Church affirmant l'indécidabilité du calcul des prédicats

[12]Rappelons qu'un atome est une forme prédicative, c'est-à-dire l'application d'un prédicat d'arité n à n arguments qui sont des termes; une formule est un atome si et seulement si elle ne contient ni connecteur ni quantificateur. Un littéral est un atome (littéral positif) ou la négation d'un atome (littéral négatif). Une clause est une disjonction de littéraux; elle est unitaire si elle comporte un seul littéral. Une clause de Horn est une clause ne comportant aucun littéral positif (clause de Horn négative) ou exactement un littéral positif (clause de Horn définie). Un cube est une conjonction de littéraux; un cube positif est une conjonction de littéraux positifs, c'est-à-dire d'atomes.

(cf. § I.8.6.2), que des limitations sont inévitables; en particulier, l'algorithme peut ne pas se terminer si la conclusion n'est pas conséquence logique des hypothèses.[13] Par ailleurs, la programmation logique fournit une preuve constructive, en ce sens que, si la fermeture existentielle du cube-conclusion est reconnue comme conséquence logique des hypothèses, alors l'algorithme doit aussi fournir des instances appropriées des variables intervenant dans ce cube (en fait, ce sont ces instances qui intéressent l'utilisateur et qui constituent les résultats fournis par l'exécution d'un programme logique).

Les quelques exemples de programmes introduits jusqu'ici, quoique très simples, suffisent à montrer que PROLOG n'est pas une réalisation entièrement satisfaisante du concept de programmation logique. Nous avons vu, notamment, qu'un programme PROLOG est une liste de clauses et non un ensemble; l'ordre des clauses a un impact parfois important sur le comportement du programme, de même que l'ordre des littéraux au sein des clauses. De même, nous avons vu que certaines exécutions de programmes PROLOG pouvaient ne pas se terminer. Ces phénomènes et d'autres du même genre justifieraient une étude générale, théorique, des liens entre une programmation logique idéale d'une part, et les systèmes PROLOG réels d'autre part. Cependant, une telle étude sortirait largement du cadre de cet ouvrage (cf. [Lloyd 87] ou [Doets 94]. Dans la suite de ce paragraphe, nous donnons quelques points saillants de la théorie de la programmation logique et de PROLOG. Le cas propositionnel étant nettement plus simple que le cas général prédicatif, nous le développerons davantage.

11.3.1. *La programmation logique propositionnelle*

Le problème de la programmation logique propositionnelle consiste à déterminer si une proposition est ou n'est pas conséquence logique d'un ensemble de clauses de Horn définies. Une variante consiste à déterminer l'ensemble des propositions conséquences logiques d'un ensemble de clauses de Horn définies. Rappelons d'abord brièvement la solution à ces problèmes introduite au volume précédent (cf. § II.5.6), qui repose sur l'algorithme de résolution unitaire (positive) représenté à la figure 11.2.

Cet algorithme permet de tester assez efficacement l'inconsistance d'un ensemble S_0 de clauses de Horn propositionnelles.[14] L'algorithme comporte un composant non déterministe de choix. Si plusieurs couples (p, c) peuvent être sélectionnés à un moment donné, un couple arbitraire acceptable est effectivement choisi. Remarquons aussi qu'une exécution de cet algorithme peut s'arrêter *normalement*, quand la condition $\mathbf{F} \in S$ devient vraie, ou

[13] En pratique, d'autres limitations sont à accepter si l'on souhaite pour l'algorithme une vitesse d'exécution "raisonnable"; cela justifie notamment le fait que l'on se restreigne le plus souvent aux clauses de Horn.

[14] Toute clause est ici assimilée à l'ensemble des littéraux dont elle est la disjonction. La clause vide (ensemble vide de littéraux) est naturellement notée **F**.

$\{S := S_0\}$
Tant que $\mathbf{F} \notin S$ faire
choisir p et c tels que
- p est une clause unitaire positive de S,
- c est une clause de S contenant $\neg p$;
$r := c \setminus \{\neg p\}$;
$S := (S \setminus \{c\}) \cup \{r\}$.

Figure 11.2 *Résolution unitaire positive*

s'arrêter *anormalement*, si aucun choix n'est possible pour p et c. On appelle *dérivation* une exécution de l'algorithme de résolution et, comme nous l'avons vu au paragraphe I.4.4, il est commode de représenter les dérivations par des arbres. Une dérivation se terminant par la production de la clause vide est une *réfutation*. Un exemple de réfutation est donné à la figure 11.3; cet exemple prouve l'inconsistance de l'ensemble

$$S = \{t \vee \neg p \vee \neg r\,,\ p \vee \neg r \vee \neg t\,,\ r\,,\ t \vee \neg q\,,\ q\,,\ \neg p \vee \neg q \vee \neg r\}\,.$$

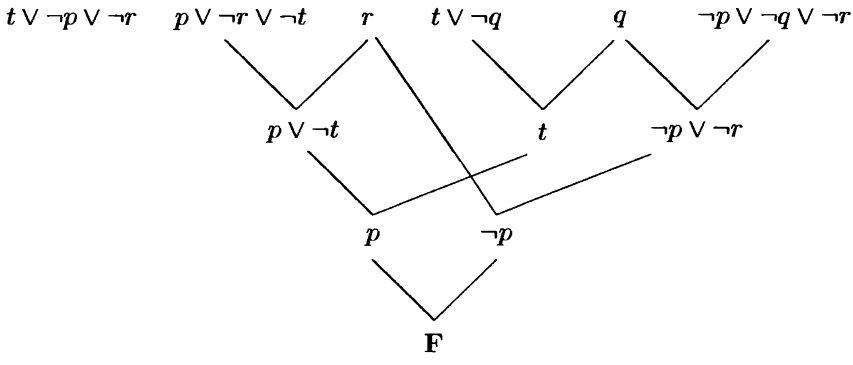


Figure 11.3 *Arbre de réfutation unitaire pour l'ensemble S*

Au paragraphe II.5.6, nous avions démontré que, pour tester si une proposition p est conséquence logique d'un programme logique L (ensemble de clauses de Horn définies), il suffit d'appliquer l'algorithme à l'ensemble $S_0 =_{def} L \cup \{\neg p\}$. L'exécution se termine toujours, normalement si $L \models p$ et anormalement sinon; le temps d'exécution est au pire fonction quadratique du nombre de littéraux intervenant dans S_0. Nous avions aussi prouvé que, si l'algorithme est appliqué à l'ensemble L (toujours consistant), l'exécution se termine toujours anormalement; l'ensemble des propositions conséquences logiques de L sont exactement les propositions qui interviennent comme clauses unitaires dans la valeur finale de l'ensemble S. Ces propositions déterminent

le modèle minimal, ou modèle canonique, du programme logique L.

Nous avions enfin noté que l'algorithme de résolution unitaire n'accorde aucun rôle particulier à la proposition p à tester; sa négation est simplement adjointe au programme logique L. L'algorithme de résolution d'entrée, qui est la version propositionnelle abstraite de l'algorithme PROLOG, permet d'exploiter le programme logique de manière plus focalisée, sans jamais perdre la question de vue (figure 11.4). Cet algorithme utilise une variable G, dont la valeur est toujours une clause de Horn négative; initialement, on a $G = G_0 = \{\neg g_1, \ldots, \neg g_n\}$, où $g_1, \ldots, g_n$ est l'ensemble conjonctif des questions.

$\{G = G_0\}$
Tant que $G \neq \mathbf{F}$ faire
choisir p et c tels que
- $\neg p \in G$,
- $c \in L$ et $p \in c$;
$G := (G \setminus \{\neg p\}) \cup (c \setminus \{p\})$.

Figure 11.4 *Résolution d'entrée*

Naturellement, on ne peut envisager de remplacer la résolution unitaire par la résolution d'entrée que si la seconde jouit des mêmes propriétés que la première. Avant d'étudier cette question, il convient d'observer que les exécutions de l'algorithme de résolution d'entrée, appelées aussi dérivations ou réfutations, peuvent aussi se représenter de manière arborescente. A titre d'exemple, nous montrons à la figure 11.5 que la proposition p est bien conséquence logique du programme logique

$$L = \{\, t \vee \neg p \vee \neg r\,,\ p \vee \neg r \vee \neg t\,,\ r\,,\ t \vee \neg q\,,\ q \,\}\,.$$

On voit que, dans un arbre de réfutation d'entrée, il existe une branche principale unissant le but initial (ici, $\neg p$) à la clause vide. Les branches auxiliaires sont de longueur 1 et unissent une clause d'entrée (d'où le nom de l'algorithme) à un but intermédiaire.

Le résultat suivant garantit que l'algorithme de résolution d'entrée hérite des propriétés importantes de l'algorithme de résolution unitaire. En particulier, il est adéquat et complet: on peut obtenir une réfutation d'entrée de $\neg p$ par rapport au programme logique L si et seulement si p est conséquence logique de L.

Théorème 11.1. L'ensemble $L \cup \{\neg p\}$ admet une réfutation unitaire si et seulement s'il admet une réfutation d'entrée basée sur $\neg p$.

Remarque préliminaire. On montrera comment transformer une réfutation unitaire en réfutation d'entrée; la transformation inverse est laissée au lecteur. La preuve se fait par induction sur le lexique, c'est-à-dire sur l'ensemble des propositions contenues dans le programme logique. On peut aussi se limiter

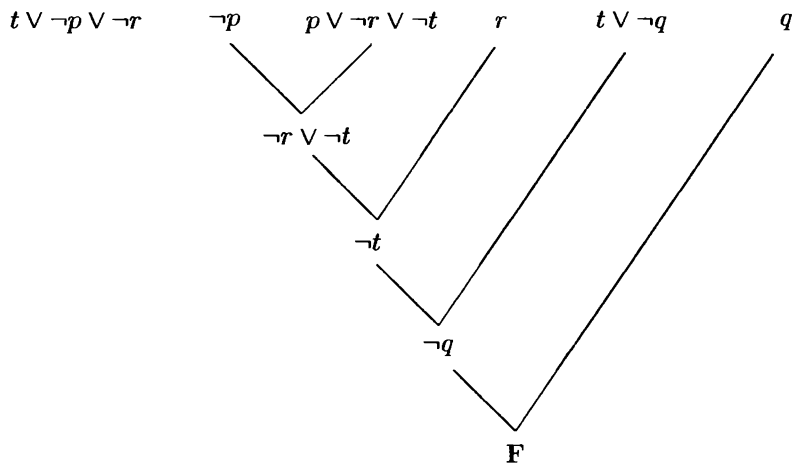


Figure 11.5 *Arbre de réfutation d'entrée pour L et p*

aux ensembles $L \cup \{\neg p\}$ minimaux, c'est-à-dire tels que tout sous-ensemble propre est consistant (et donc ne donne jamais lieu à une réfutation). En effet, tout ensemble admet un sous-ensemble minimal, et une réfutation sur le sous-ensemble minimal est aussi une réfutation sur l'ensemble original.

Preuve. Le cas de base est celui où le lexique se réduit à $\Pi = \{p\}$. L'ensemble minimal $L \cup \{\neg p\}$ est nécessairement $\{p, \neg p\}$; il admet une seule réfutation, qui est à la fois unitaire et d'entrée. Supposons maintenant que l'ensemble $L \cup \{\neg p\}$ soit minimal, que le lexique ne se réduise pas à la proposition p et qu'une réfutation unitaire existe. Cela signifie que L comporte au moins une clause unitaire positive q distincte de p; toute autre occurrence de q dans L est nécessairement sous la forme du littéral négatif $\neg q$. On forme alors l'ensemble L' obtenu à partir de L en supprimant la clause q et, dans toutes les clauses qui le contiennent, le littéral $\neg q$. Soit $L'' \subset L'$ tel que $L'' \cup \{\neg p\}$ soit minimal. La réfutation unitaire relative à $L \cup \{\neg p\}$ se transforme aisément en une réfutation unitaire de $L'' \cup \{\neg p\}$, par simple omission des étapes utilisant la clause q et par suppression de toutes les occurrences du littéral $\neg q$. Le lexique de L'' est un sous-ensemble propre du lexique de L d'où, par hypothèse d'induction, il existe une réfutation d'entrée de $L'' \cup \{\neg p\}$ basée sur $\neg p$. En rajoutant le littéral $\neg q$ aux endroits adéquats de cette réfutation, on obtient une dérivation d'entrée basée sur $\neg p$ dont la racine est $\neg q$ et dont les feuilles appartiennent à $L \cup \{\neg p\}$. Comme q est une clause de L, on complète cette dérivation par une dernière étape utilisant cette clause, ce qui conduit à une réfutation d'entrée de $L \cup \{\neg p\}$ basée sur $\neg p$.

11.3.2. PROLOG *propositionnel*

L'algorithme de PROLOG est une version concrète de l'algorithme de résolution d'entrée. Les clauses sont représentées par des listes de littéraux et le programme logique L est une liste de clauses. Les choix de p et c sont imposés par une stratégie très simple; p est nécessairement la proposition correspondant au premier littéral du but courant et c est la première clause de L convenable qui n'a pas encore été utilisée. En effet, le système PROLOG essaiera, dans l'ordre où elles se présentent dans la liste L, toutes les clauses dont la tête est p.

11.4. Programmer en PROLOG

Avant de considérer plus avant l'intérêt de la logique et de PROLOG en intelligence artificielle, il est intéressant d'écrire quelques petits programmes; ils suffisent à mettre en évidence certains principes et surtout la concision et le grand pouvoir d'expression de PROLOG: des problèmes d'apparence complexe peuvent être résolus simplement. On voit aussi que la programmation en PROLOG, simple dans son principe, requiert une grande attention; deux programmes équivalents en apparence peuvent se comporter très différemment.

11.4.1. *Listes, préfixes, suffixes, sous-listes et sous-ensembles*

Nous continuons ici l'étude des listes et des divers concepts associés. Au lieu de dériver le prédicat `prefix` du prédicat `append`, on peut le définir directement, par une clause de base et une clause récursive. D'une part, la liste vide est un préfixe de toute liste et, d'autre part, `[X|Xs]` est un préfixe de `[X|Ys]` si `Xs` est un préfixe de `Ys`. La notion de suffixe se définit de manière analogue. Enfin, on définit une sous-liste de `Xs` comme une liste d'éléments consécutifs de `Xs`. On voit immédiatement qu'une sous-liste est le suffixe d'un préfixe de la liste de départ, ce qui permet une programmation très simple. A première vue, les deux variantes `sublist1` et `sublist2` de la figure 11.6 sont équivalentes.

Nous utilisons d'abord `sublist2` pour construire les sous-listes de la liste `[a,b]`. L'arbre de recherche correspondant est représenté à la figure 11.7. Nous observons que toutes les sous-listes sont obtenues mais la sous-liste vide est construite trois fois. PROLOG fournit les résultats à l'utilisateur dès que ceux-ci sont obtenus; cela signifie que, durant la construction de gauche à droite de l'arbre de recherche, l'utilisateur reçoit successivement les solutions suivantes:

```
[], [a], [], [a,b], [b], [].
```

Si l'utilisateur réclame une solution supplémentaire, le système affiche "NO" et l'exécution s'arrête. Du point de vue de la logique, ce comportement est parfaitement correct. Par analogie avec le vocabulaire employé pour les

```
suffix(Xs,Xs).
suffix(Xs,[Y|Ys]) :- suffix(Xs,Ys).

prefix([],Xs).
prefix([X|Xs],[X|Ys]) :- prefix(Xs,Ys).

sublist1(Xs,Ys) :- suffix(Xs,Zs), prefix(Zs,Ys).

sublist2(Xs,Ys) :- prefix(Zs,Ys), suffix(Xs,Zs).
```

Figure 11.6 *Deux variantes du prédicat* `sublist`

systèmes axiomatiques, on peut dire que le programme est adéquat (seules des réponses correctes sont fournies) et complet (toutes les réponses correctes sont fournies). Néanmoins, d'un point de vue pragmatique, on préférerait que chaque résultat soit fourni une seule fois.[15]

Nous utilisons maintenant la variante `sublist1` pour vérifier si `[a]` est une sous-liste de `[b]`. Nous observons que toutes les branches finies de l'arbre de recherche (figure 11.8) se terminent par "`NO`", ce qui est normal. Par contre, on observe la présence d'une branche infinie. Ce phénomène est gênant: PROLOG ne pourra répondre à la question de l'utilisateur, puisqu'une réponse négative n'est logiquement et opérationnellement possible que quand la construction de l'arbre de recherche est achevée et n'a pas produit de réponse positive. En fait, la variante `sublist1` est inadéquate dans ce contexte même si, à proprement parler, elle n'est pas incorrecte. Un autre problème survient si `sublist1` est utilisé pour construire toutes les sous-listes d'une liste donnée. PROLOG donne bien toutes les réponses (pas dans le même ordre qu'avec le programme `sublist2`),[16] mais, et cela est gênant, PROLOG n'arrête pas son exécution; une branche infinie et inutile se trouve à la droite de l'arbre de recherche. Curieusement, `sublist2` fonctionne mieux, au moins pour vérifier

[15] On peut écrire une variante remplissant cette exigence. Les codes pour `prefix` et `suffix` ne sont pas modifiés mais on définit `sublist3` par les deux clauses suivantes:

```
sublist3([],Xs).
sublist3([X|Xs],Ys) :- prefix(Zs,Ys), suffix([X|Xs],Zs).
```

Les solutions sont produites et fournies à l'utilisateur dans un ordre précis; par exemple, les solutions de:

```
sublist3(Xs,[a,b,c,d]).
```

sont `[]`, `[a]`, `[a,b]`, `[b]`, `[a,b,c]`, `[b,c]`, `[c]`, `[a,b,c,d]`, `[b,c,d]`, `[c,d]`, `[d]`.

[16] Les sous-listes de `[a,b,c]` sont fournies par `sublist1` dans l'ordre:
`[]`, `[a]`, `[a,b]`, `[a,b,c]`, `[]`, `[b]`, `[b,c]`, `[]`, `[c]`, `[]`;
alors que, par `sublist2`, elles sont fournies dans l'ordre:
`[]`, `[a]`, `[]`, `[a,b]`, `[b]`, `[]`, `[a,b,c]`, `[b,c]`, `[c]`, `[]`.

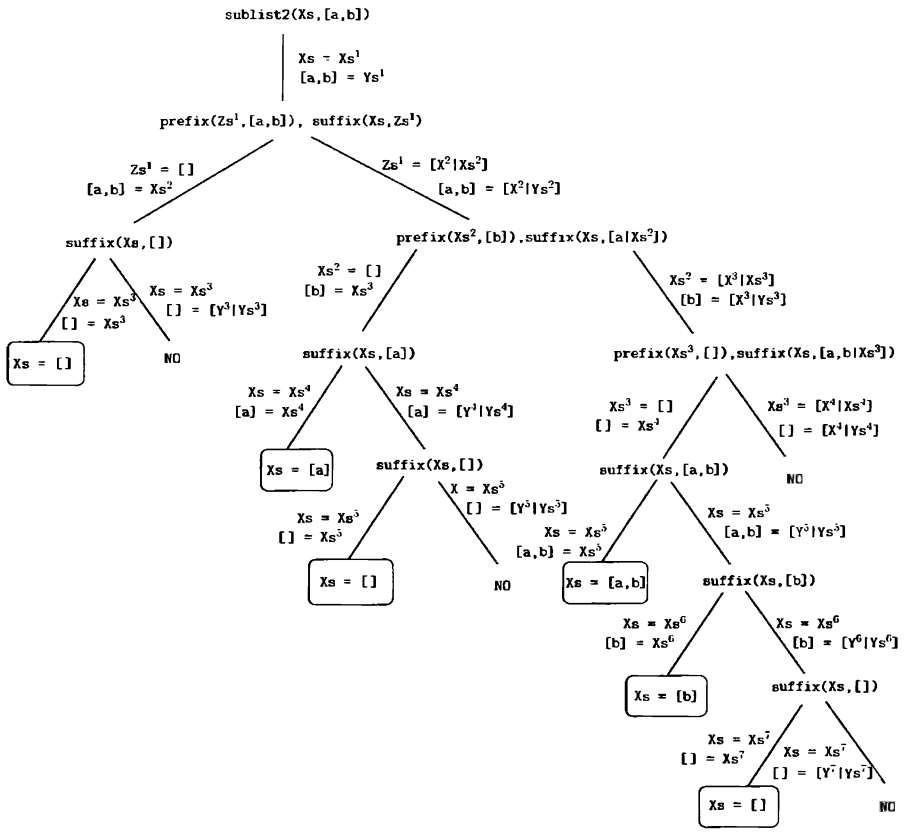


Figure 11.7 *Un arbre de recherche fini pour un calcul de sous-listes*

que [a] n'est pas une sous-liste de [b], ainsi que le montre l'arbre de recherche de la figure 11.9. Cet arbre est très différent du précédent, et fini. Une première conclusion est que l'ordre des éléments dans le corps d'une clause PROLOG peut être très important.

Le fait que le comportement de PROLOG dépende de l'ordre des prédicats au sein des corps des clauses pourrait paraître surprenant; en effet, d'après la sémantique déclarative, il ne devrait pas y avoir de différence entre les deux clauses "`A :- B,C.`" et "`A :- C,B.`". L'explication est que PROLOG tient pour vraies seulement les propositions démontrables au moyen d'une stratégie spécifique, et cette stratégie n'est pas nécessairement complète. Notons aussi que, du point de vue déclaratif, l'ordre des clauses elles-mêmes dans le programme devrait être indifférent. En fait, l'ordre des clauses détermine celui des branches des arbres de recherche et influe donc sur l'ordre dans lequel PROLOG produit et fournit ses réponses. Cela peut devenir gênant si

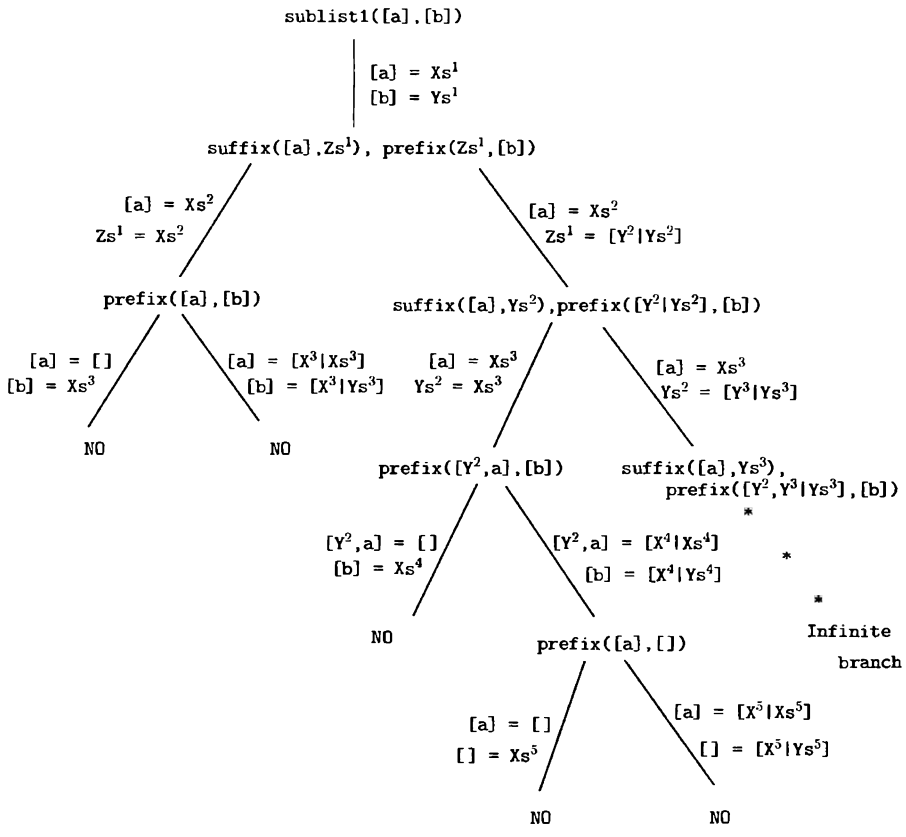


Figure 11.8 *Un arbre de recherche infini pour un calcul de sous-listes*

l'arbre comporte une ou plusieurs branches infinies; seule la partie de l'arbre se trouvant à gauche de la première branche infinie sera explorée, ce qui peut induire la perte de certaines, voire de toutes les réponses déclarativement attendues. Différents exemples de ce phénomène seront rencontrés dans la suite.

On utilise souvent des listes pour représenter d'autres structures de données. Par exemple, il est fréquent de modéliser un ensemble par une liste sans répétition, ou encore par une liste triée sans répétition.[17] Il est facile d'écrire un prédicat `subset_1` qui reconnaît et/ou construit les sous-ensembles d'un

[17]Le tri alourdit la construction de la structure mais peut alléger son exploration; de plus, il permet d'obtenir une propriété intéressante: chaque ensemble est représenté par une et une seule liste triée sans répétition.

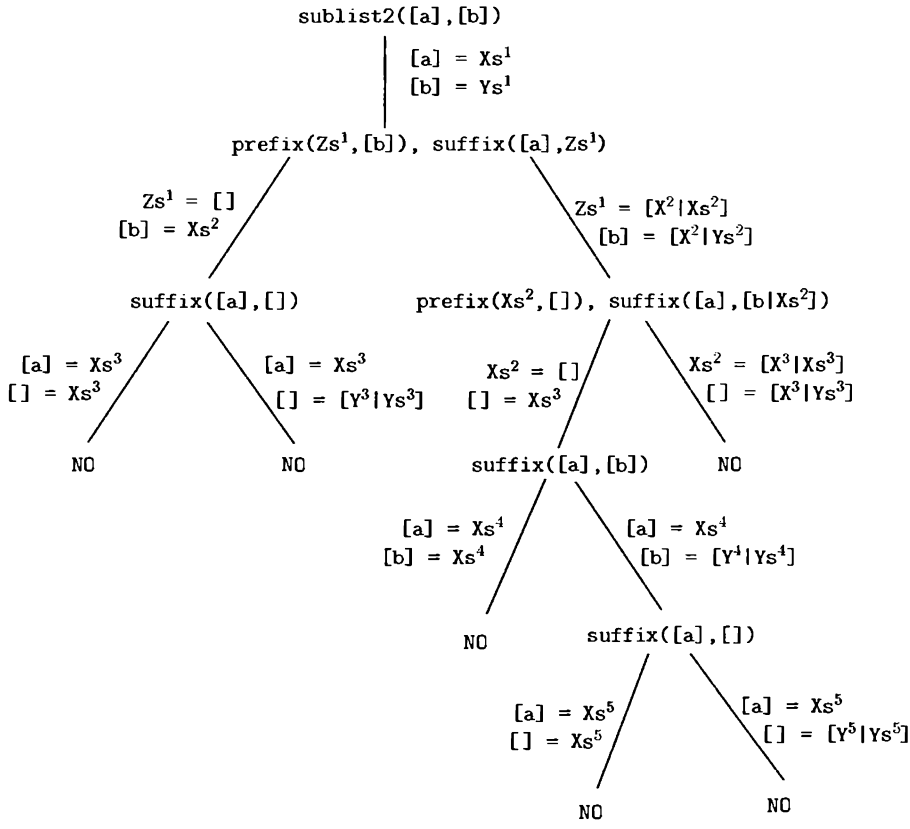


Figure 11.9 *Un arbre de recherche fini infructueux pour un calcul de sous-listes*

ensemble :

```
subset_1([],[]).
subset_1([X|Xs],[X|Zs]) :- subset_1(Xs,Zs).
subset_1(Xs,[X|Zs]) :- subset_1(Xs,Zs).
```

Ce prédicat construit correctement tous les sous-ensembles; par exemple, on a :

```
?- subset_1(Xs,[a,b,c]).
Xs = [a, b, c] ;
Xs = [a, b] ;
Xs = [a, c] ;
Xs = [a] ;
Xs = [b, c] ;
```

```
Xs = [b] ;
Xs = [c] ;
Xs = [] ;
No
```

Il peut aussi être utilisé pour reconnaître les sous-ensembles mais il ne reconnaîtra pas les listes avec répétitions et/ou inversions, comme le montre la session suivante :

```
?- subset_1([1,3],[1,2,3,4]).
Yes
?- subset_1([1,1,3],[1,2,3,4]).
No
?- subset_1([3,1],[1,2,3,4]).
No
```

Si on souhaite que les répétitions et inversions soient reconnues, on pourra utiliser

```
member(X,[X|Xs]).
member(X,[Y|Xs]) :- member(X,Xs).

subset_2([],Xs).
subset_2([Y|Ys],Xs):- member(Y,Xs), subset_2(Ys,Xs).
```

On a alors :

```
?- subset_2([1,1,3],[1,2,3,4]).
Yes

?- subset_2([3,1],[1,2,3,4]).
Yes
```

Par contre, `subset_2` ne peut pas être utilisé pour la génération des sous-ensembles :

```
?- subset_2(Xs,[1,2,3,4]).
Xs = [] ;
Xs = [1] ;
Xs = [1, 1] ;
Xs = [1, 1, 1] ;
...
```

Remarque. Le prédicat auxiliaire `member` est très utile; il sera abondamment employé au chapitre suivant.

Le prédicat `subset_1` construit les sous-ensembles un à un mais on programme aisément le prédicat `subset_list` qui permet la construction de la liste complète des sous-ensembles d'un ensemble donné :

```
subset_list([],[[]]).
subset_list([X|Xs],Zss) :- subset_list(Xs,Uss),
```

```
                                put_in_all(X,Uss,Vss),
                                append(Vss,Uss,Zss).
put_in_all(A,[],[]).
put_in_all(A,[Bs|Bss],[[A|Bs]|Css]) :- put_in_all(A,Bss,Css).
```

L'exemple suivant montre comment ce prédicat et le prédicat auxiliaire `put_in_all` fonctionnent:

```
?- put_in_all(x,[[a,b],[],[c,d,e]],Xss).
Xss = [[x, a, b], [x], [x, c, d, e]]
?- subset_list([a,b,c],Xss).
Xss = [[a, b, c], [a, b], [a, c], [a], [b, c], [b], [c], []]
```

11.4.2. *Machines abstraites et fonctions récursives*

Les mécanismes PROLOG que nous venons de décrire et d'illustrer peuvent sembler rudimentaires mais cette apparence est trompeuse; ils permettent en fait de programmer tout ce qui est programmable, bien souvent de manière simple et concise (mais pas nécessairement très efficace). Nous démontrons ceci en recréant en PROLOG les machines abstraites, y compris la machine de Turing.[18] Nous montrons aussi comment les fonctions récursives introduites par Gödel (cf. § I.2.3.4) peuvent être simulées en PROLOG.

Remarque. Nous ne voulons pas introduire ou rappeler ici la théorie des machines abstraites, qui est une branche relativement distincte de la logique. Toutefois, les exemples donnés dans cette section ne requièrent pas de connaissance préalable de cette théorie.

11.4.2.1. *Automates*

Les automates finis ou machines à états finis[19] constituent un mécanisme de calcul très élémentaire, ce qui ne les empêchent pas d'avoir de nombreuses applications, tant en informatique théorique qu'en programmation réelle. L'automate fini représenté à la figure 11.10 comporte les deux états q_0 (l'état initial, repéré par une grande pointe de flèche) et l'état q_1. (L'état q_0 est aussi final, ce qu'indique le double cercle.) Cet automate reconnaît certains mots construits au moyen de l'alphabet $\{a, b\}$.[20] Lire a dans l'état q_0 conduit en l'état q_1 et lire b en l'état q_1 conduit en l'état q_0. On en déduit immédiatement que le langage reconnu par l'automate, c'est-à-dire l'ensemble des mots acceptés par cet automate, est:

[18] Elles sont analogues aux machines de Post décrites au paragraphe I.8.5.2.

[19] Plus précisément: les machines dont l'espace des états est un ensemble fini.

[20] Les mots sont simplement des suites de lettres de l'alphabet. Un langage est un ensemble de mots. Il existe une infinité non dénombrable de langages sur l'alphabet $\{a, b\}$; les automates et autres machines abstraites permettent de reconnaître et de construire les langages les plus intéressants.

$$(ab)^* =_{def} \{\varepsilon, ab, abab, ababab, \ldots\},$$

où ε désigne le mot vide.

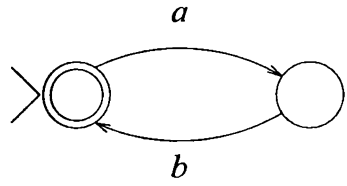


Figure 11.10 *Un automate fini acceptant le langage* $(ab)^*$

Cet automate se représente facilement en PROLOG par quatre clauses qui sont des faits fondamentaux. De plus, le comportement de cet automate (ou de n'importe quel automate fini) est simulé par un interpréteur PROLOG composé de trois clauses élémentaires (voir figure 11.11).

```
/* Version Prolog d'un automate acceptant le langage (ab)* */
init(q0).
fin(q0).
delta(q0,a,q1).
delta(q1,b,q0).
/* Interprétation d'un automate représenté en Prolog */
acc(Xs) :- init(Q), acc(Q,Xs).
acc(Q,[]) :- fin(Q).
acc(Q,[X|Xs]) :- delta(Q,X,Q1), acc(Q1,Xs).
```

Figure 11.11 *Automate fini, représentation et simulation en* PROLOG

Les automates finis, quoique très simples, permettent de résoudre des problèmes intéressants et de reconnaître une vaste classe (dénombrable) de langages, dits réguliers. Il existe toutefois des langages non réguliers très importants, comme le langage des palindromes, c'est-à-dire des mots identiques à leur "miroir", tels ε, b, bb, aba, $bbabab$, etc). Le langage des palindromes appartient à la famille des langages hors-contexte, qui est un sur-ensemble de la famille des langages réguliers. Les langages hors-contexte sont exactement ceux reconnus par les automates à pile non déterministes. Ces automates sont obtenus en ajoutant aux automates finis une mémoire non bornée, structurée comme une pile ("premier entré, dernier sorti"). Chaque transition de l'automate à pile peut provoquer l'ajout d'une lettre sur la pile, ou le retrait d'une lettre. Un équivalent PROLOG d'un automate à pile reconnaisseur du langage des palindromes sur l'alphabet $\{a, b\}$ est illustré à la figure 11.12.[21] On

[21] La lecture de la partie gauche du palindrome correspond à une succession de transitions $q_0 \rightarrow q_0$ (première clause); toute lettre lue est aussi empilée. L'automate doit détecter le

observe que l'interpréteur PROLOG pour les automates à pile est très semblable à celui utilisé plus haut pour les automates finis.

```
/* Automate à pile pour le langage des palindromes sur {a,b} */
init(q0).
fin(q1).
delta(q0,X,S,q0,[X|S]).
delta(q0,X,S,q1,[X|S]).
delta(q0,X,S,q1,S).
delta(q1,X,[X|S],q1,S).
/* Interprétation */
acc(Xs) :- init(Q), acc(Q,Xs,[]).
acc(Q,[],[]) :- fin(Q).
acc(Q,[X|Xs],S) :- delta(Q,X,S,Q1,S1), acc(Q1,Xs,S1).
```

Figure 11.12 *Automate à pile, représentation et simulation*

Les automates à pile sont plus expressifs que les automates finis, surtout si, comme ici, le non-déterminisme est autorisé. Néanmoins, certains langages restent hors de portée des automates à pile, ce qui signifie qu'aucun automate à pile ne peut reconnaître exactement ces langages-là. Par exemple, il est facile de concevoir un automate à pile acceptant le langage:

$$\bigcup_{n\in\mathbb{N}} a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\},$$

mais aucun automate à pile ne peut reconnaître le langage:

$$\bigcup_{n\in\mathbb{N}} a^n b^n c^n = \{\varepsilon, abc, aabbcc, aaabbbccc, \ldots\},$$

défini sur l'alphabet $\{a, b, c\}$. Ceci est dû à la restriction qui pèse sur le mode d'accès aux éléments d'une pile: on doit nécessairement retirer le dernier élément placé sur la pile avant d'accéder aux autres éléments. Cette restriction disparaît si l'on dispose d'une seconde pile, où les éléments retirés de la première pile peuvent être entreposés,[22] et les automates à deux piles ont la même puissance d'expression que les machines de Turing ou de Post (cf. I.8.5.2). Une machine de Turing est un automate fini augmenté d'une mémoire non bornée organisée en un ruban. Le mode d'accès aux éléments d'un ruban est séquentiel, mais il n'est pas nécessaire de retirer un élément avant d'accéder

milieu du palindrome (choix non déterministe), ce qui le conduit en l'état q_1, (deuxième δ-clause pour un palindrome pair, troisième pour un palindrome impair). L'automate consomme ensuite la partie droite du palindrome, en vidant la pile et en vérifiant l'égalité de la lettre lue avec la lettre dépilée (quatrième δ-clause).

[22] A titre d'exercice, le lecteur peut concevoir un automate à une pile pour le langage $\{a^n b^n : n \in \mathbb{N}\}$ et un automate à deux piles pour le langage $\{a^n b^n c^n : n \in \mathbb{N}\}$.

au suivant. La figure 11.14 représente une machine de Turing reconnaissant le langage :

$$\bigcup_{n\in\mathbb{N}} a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\},$$

ainsi qu'un interpréteur PROLOG pour ce type de machines abstraites. Le terme `config(Q,Ds,Head,Fs)` représente la configuration dont l'état est `Q`, la partie gauche du ruban est la liste `Ds` inversée, la cellule-mémoire repérée par la tête de lecture est `Head` et la partie droite du ruban est `Fs`. La machine utilise des symboles auxiliaires pendant le calcul.

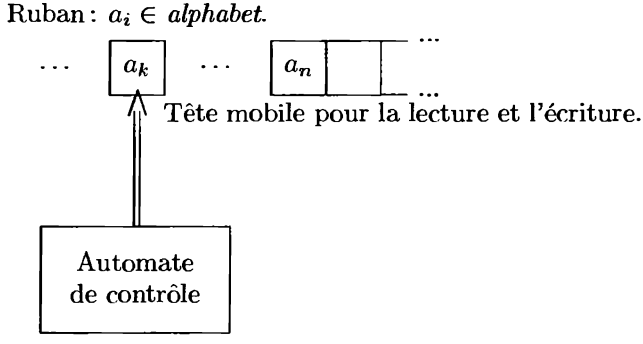


Figure 11.13 *Architecture d'une machine de Turing*

11.4.2.2. *Fonctions récursives*

Les machines de Turing constituent un outil théorique précieux mais ne conviennent guère pour effectuer des calculs réels. Les fonctions récursives, introduites par Gödel pour des raisons également théoriques, sont en apparence plus proches du monde numérique. Nous rappelons ici leur définition et montrons que, une fois de plus, la simulation de ces fonctions en PROLOG est immédiate. Il existe trois familles de fonctions récursives de base, à savoir les fonctions nulles, la fonction successeur et les projections. Elles sont résumées dans le tableau suivant.

$(x_1, \ldots, x_n) \mapsto 0$: fonction zéro d'arité n;

$x \mapsto x + 1$: fonction successeur;

$(x_1, \ldots, x_n) \mapsto x_i$: $i$ème projection d'arité n.

On obtient des fonctions récursives plus complexes en utilisant trois mécanismes de combinaison, qui sont la composition, la récursion primitive et la minimisation. La composition permet de définir une fonction f en termes

```
/* Exemple de machine de Turing codée en Prolog */
initial(q0).                final(q4).

delta(q0,a,q1,x,right).  delta(q0,y,q3,y,right).
delta(q1,a,q1,a,right).  delta(q1,b,q2,y,left).
delta(q1,y,q1,y,right).  delta(q2,a,q2,a,left).
delta(q2,x,q0,x,right).  delta(q2,y,q2,y,left).
delta(q3,y,q3,y,right).  delta(q3,blanc,q4,blanc,right).

/* Interpréteur Prolog pour machines de Turing */
accept_MT([]) :- initial(Q), accept(config(Q,[],blanc,[])).
accept_MT([A|As]) :- initial(Q), accept(config(Q,[],A,As)).

accept(config(Q,Ds,Head,Fs)) :- final(Q).
accept(config(Q,Ds,Head,Fs)) :-
   next_config(config(Q,Ds,Head,Fs), config(Qn,Dsn,Headn,Fsn)),
   accept(config(Qn,Dsn,Headn,Fsn)).

next_config(config(Q,Ds,Head,Fs), config(Qn,Dsn,Headn,Fsn)) :-
   delta(Q,Head,Qn,Symb,Act),
   modif(config(Q,Ds,Head,Fs), Symb, Act, config(Qn,Dsn,Headn,Fsn)).

modif(config(Q,Ds,Head,[]),Symb,right,config(Qn,[Symb|Ds],blanc,[])).
modif(config(Q,Ds,Head,[F|Fs]),Symb,right,config(Qn,[Symb|Ds],F,Fs)).
modif(config(Q,[D|Ds],Head,Fs),Symb,left,config(Qn,Ds,D,[Symb|Fs])).
```

Figure 11.14 *Machine de Turing et interpréteur*

de fonctions récursives $g_1, \ldots, g_m$ et h définies au préalable, selon le schéma suivant :

$$f(x_1, \ldots, x_n) =_{def} h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)) .$$

La récursion primitive permet de définir une fonction f en termes de fonctions récursives g et h définies au préalable, selon le schéma suivant :

$$f(x_1, \ldots, x_n, 0) = h(x_1, \ldots, x_n) ,$$
$$f(x_1, \ldots, x_n, y+1) = g(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)) .$$

La minimisation permet de définir une fonction f en termes de la fonction récursive g définie au préalable, selon le schéma suivant :

$$f(x_1, \ldots, x_n) = \mu y.[g(x_1, \ldots, x_n, y) = 0] ,$$

où $\mu y.p(y)$ désigne le plus petit entier naturel y, s'il existe, tel que $p(y)$ soit vrai. Le schéma de minimisation permet la définition de fonctions partielles, contrairement aux deux autres schémas. Les fonctions obtenues sans l'aide du schéma de minimisation sont les fonctions primitives récursives; elles sont toujours totales.

Le langage PROLOG pur (au contraire des systèmes PROLOG réels) ne possède pas de primitives arithmétiques, puisque l'unique type de donnée est le

terme. On peut néanmoins représenter les nombres par des termes particuliers et convenir, par exemple, que la suite des entiers naturels $0, 1, 2, 3, \ldots$ est représentée par la suite des termes `o`, `s(o)`, `s(s(o))`, `s(s(s(o)))`, ... Nous convenons aussi de représenter une fonction f d'arité n par un prédicat `f_` d'arité $n+1$, le fait `f_(X1, ... ,Xn,Y)` représentant l'égalité $f(x_1, \ldots, x_n) = y$. A la fonction nulle d'arité n est donc associé le prédicat `zero_`, défini par le fait :

```
zero_(X1,...,Xn,o).
```

On définit de même le prédicat associé à la $j$ième projection d'arité n :

```
pr_j(X1,...,Xn,Xj).
```

A la fonction successeur correspond un prédicat binaire :

```
succ_(X,s(X)).
```

La clause suivante modélise le mécanisme de composition :

```
f_(X1,...,Xn,Y) :-
      g_1(X1,...,Xn,Y1),...,g_m(X1,...,Xn,Ym),
      h_(Y1,...,Ym,Y).
```

Deux clauses permettent de simuler la récursion primitive :

```
f_(X1,...,Xn,o,Z) :- h_(X1,...,Xn,Z).
f_(X1,...,Xn,s(Y),Z) :- f_(X1,...,Xn,Y,U),
                        g_(X1,...,Xn,Y,U,Z).
```

Enfin, voici le programme de trois clauses réalisant la minimisation :

```
f_(X1,...,Xn,Y) :- g_(X1,...,Xn,o,U),r_(X1,...,Xn,o,U,Y).
r_(X1,...,Xn,Y,o,Y).
r_(X1,...,Xn,Y,s(V),Z) :- g_(X1,...,Xn,s(Y),U),
                          r_(X1,...,Xn,s(Y),U,Z).
```

Le prédicat `r_` à $(n+3)$ arguments est associé à une fonction auxiliaire r d'arité $(n+2)$ qui peut être spécifiée comme suit :

Si $g(x_1, \ldots, x_n, \xi) > 0$ *pour tout* $\xi = 0, 1, \ldots, y-1$
et si $g(x_1, \ldots, x_n, y) = u$,
alors $r(x_1, \ldots, x_n, y, u) = \inf\{\zeta : \zeta \geq y \wedge g(x_1, \ldots, x_n, \zeta) = 0\}$.

On voit que la valeur $r(x_1, \ldots, x_n, y, u)$ n'est pas définie si $u > 0$ et $g(x_1, \ldots, x_n, \zeta) > 0$ pour tout $\zeta > y$. Quand c'est le cas, l'évaluation de la question `r_(X1,...,Xn,Y,U,Z)` ne se termine pas.

Il est facile de programmer dans ce cadre les fonctions arithmétiques habituelles; on a notamment :

```
add(X,o,X).
add(X,s(Y),s(Z)) :- add(X,Y,Z).

mult(X,o,o).
mult(X,s(Y),U) :- mult(X,Y,Z), add(Z,X,U).
```

Ces programmes ne constituent naturellement pas un moyen très efficace et très commode de calculer numériquement, mais il est intéressant d'observer que les mécanismes de base de PROLOG, en dépit de leur simplicité, permettent de calculer tout ce qui est calculable.

11.4.3. *Calcul symbolique*

PROLOG n'est pas vraiment adapté au calcul numérique mais convient très bien pour le calcul symbolique. Pour illustrer ce fait, nous considérons une application classique, la transformation d'expressions arithmétiques ordinaires en expressions préfixées ou postfixées ("notation polonaise", directe ou inverse). Pour simplifier le problème, nous supposons que les opérations arithmétiques sont binaires et que les parenthèses ne sont jamais omises. Une expression arithmétique, telle que :

$$((2 * (9 - (3 + 4))) + 5),$$

est en fait une notation linéaire désignant un arbre dont chaque nœud interne est un opérateur; les deux fils d'un opérateur sont naturellement les opérandes. L'arbre correspondant à l'expression ci-dessus est représenté à la figure 11.15. Comme nous n'avons pas introduit les primitives PROLOG

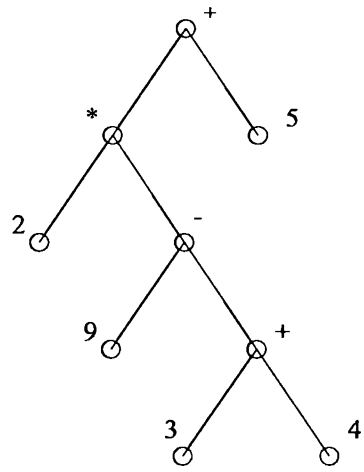


Figure 11.15 *Une expression arithmétique*

permettant la manipulation de termes quelconques, nous supposons qu'une expression arithmétique est représentée par un $\mathbb{N}$-arbre. Un $\mathbb{N}$-arbre atomique

est simplement un nombre.[23] Un $\mathbb{N}$-arbre composé est une liste à trois éléments `[X,Y,Z]` où `X` est un opérateur arithmétique[24] et `Y` et `Z` sont les $\mathbb{N}$-arbres représentant les opérandes de gauche et de droite, respectivement. Le $\mathbb{N}$-arbre associé à l'expression arithmétique de la figure 11.15 est donc :

```
[+,[*,2,[-,9,[+,3,4]]],5].
```

La liste des symboles correspondant à la forme usuelle, infixée de cette expression est :

```
[(,(,2,*,(,9,-,(,3,+,4,),),),+,5,)].
```

Cette forme semble commode mais la notation infixée a l'inconvénient de requérir des parenthèses, ce que permettent d'éviter les notations préfixée et postfixée. En notation préfixée, l'opérateur apparaît avant ses opérandes, tandis qu'il apparaît après en notation postfixée.[25] Dans notre exemple, la liste des symboles de la notation préfixée est :

```
[+,*,2,-,9,+,3,4,5].
```

Pour la notation postfixée, on a :

```
[2,9,3,4,+,-,*,5,+].
```

Un problème se posant naturellement est celui de la conversion d'un $\mathbb{N}$-arbre quelconque en la liste des symboles correspondant à l'une des trois notations. Pour la forme infixée, nous aurons des parenthèses; comme il s'agit de caractères spéciaux en PROLOG, nous les remplacerons par des lettres, `l` pour une parenthèse gauche et `r` pour une parenthèse droite. La figure 11.16 présente une première solution à ce problème. Cette solution est correcte mais

```
tree_to_pref(T,[T]) :- number(T).
tree_to_pref([X,Y,Z],[X|Xs]) :-
    tree_to_pref(Y,Ys), tree_to_pref(Z,Zs), append(Ys,Zs,Xs).
tree_to_postf(T,[T]) :- number(T).
tree_to_postf([X,Y,Z],Us) :-
    tree_to_postf(Y,Ys), tree_to_postf(Z,Zs),
    append(Ys,Zs,Xs), append(Xs,[X],Us).
tree_to_inf(T,[T]) :- number(T).
tree_to_inf([X,Y,Z],[l|Xs]) :-
    tree_to_inf(Y,Ys), tree_to_inf(Z,Zs),
    append([X|Zs],[r],Us), append(Ys,Us,Xs).
```

Figure 11.16 *Analyse de $\mathbb{N}$-arbre, solution naïve*

[23] En PROLOG pur, les nombres sont des termes atomiques; ils acquièrent une signification spécifiques en PROLOG réel, grâce aux opérateurs arithmétiques.

[24] toujours sans signification spécifique en PROLOG pur.

[25] La notation postfixée est utilisée dans les calculateurs de poche Hewlett-Packard.

pas très efficace; la conversion d'une expression prendra un temps proportionnel au carré de la longueur de l'expression. Cela est dû au recours systématique au prédicat append. En outre, les prédicats proposés sont "à sens unique" : ils peuvent être utilisés pour transformer un $\mathbb{N}$-arbre en une liste de symboles mais ils ne peuvent opérer la conversion inverse, pourtant tout aussi utile. Il est facile de remédier à ce dernier défaut; les programmes de conversion inverse sont représentés à la figure 11.17. Ces programmes sont encore moins efficaces

```
pref_to_tree([T],T) :- number(T).
pref_to_tree([X|Xs],[X,Y,Z]) :-
     append(Ys,Zs,Xs), pref_to_tree(Ys,Y), pref_to_tree(Zs,Z).
postf_to_tree([T],T) :- number(T).
postf_to_tree(Us,[X,Y,Z]) :-
    append(Xs,[X],Us), append(Ys,Zs,Xs),
    postf_to_tree(Ys,Y), postf_to_tree(Zs,Z).
inf_to_tree([T],T) :- number(T).
inf_to_tree([l|Xs],[X,Y,Z]) :-
    append(Ys,Us,Xs), append([X|Zs],[r],Us),
    inf_to_tree(Ys,Y), inf_to_tree(Zs,Z).
```

Figure 11.17 *Synthèse de $\mathbb{N}$-arbre, solution naïve*

que les précédents parce que la première étape de la conversion d'une liste en un $\mathbb{N}$-arbre est la coupure de la liste au moyen du prédicat append. Aucun "indice" n'est fourni au système concernant l'endroit de la liste où cette coupure doit se faire, donc le système procède par essais successifs. (On peut s'en rendre compte en construisant un arbre de recherche.) Néanmoins, si l'expression à convertir n'est pas trop longue, la bonne réponse est fournie à l'utilisateur sans délai perceptible, et surtout sans que les effets du "tâtonnement" se fassent sentir. On peut donc considérer que le système PROLOG simule un comportement intelligent puisque, du point de vue de l'utilisateur, le système opère le bon choix. La gestion du non-déterminisme est un mécanisme fondamentalement simple mais qui contribue largement à l'utilité de PROLOG en intelligence artificielle. Nous en verrons quelques exemples dans la suite.

11.5. Pour en savoir plus

De nombreux livres ont été consacrés aux principes et à la pratique de la programmation classique et de PROLOG, ainsi qu'à l'utilisation de PROLOG en intelligence artificielle. Citons notamment, en français, [Delahaye 86] et [Thayse et al. 88]. Les aspects théoriques de la programmation logique sont étudiés dans [Doets 94] et [Lloyd 87]. Parmi les ouvrages d'initiation à PROLOG et d'applications à l'intelligence artificielle, on notera [Bratko 90], [Shoam 95] et [Sterling et Shapiro 94].

Chapitre 12
Le langage PROLOG en intelligence artificielle

Nous avons déjà montré que la gestion du non-déterminisme permettait d'organiser simplement une recherche dans un espace d'états. Cette recherche consiste souvent en la construction d'un arbre ou d'un graphe dont les nœuds sont étiquetés par des états. Deux exemples significatifs sont la recherche systématique des solutions au problème des Tours de Hanoï (cf. Fig. 10.5) et celle d'une tactique optimale pour le jeu d'Hexapion (cf. Fig. 10.7).

Nous avons vu aussi que le langage PROLOG permettait une gestion simple (quoique pas toujours efficace) du non-déterminisme. L'exemple le plus typique, et peut-être aussi celui dont l'usage est le plus fréquent, est constitué par le prédicat append. En effet, celui-ci permet non seulement de concaténer deux listes, mais aussi de réaliser l'opération inverse, c'est-à-dire de décomposer une liste en un préfixe et un suffixe, de toutes les manières possibles.

Le système PROLOG permet de résoudre simplement des problèmes admettant plusieurs solutions. En intelligence artificielle, on peut fréquemment présenter un problème dans ce cadre. Par exemple, un programme de jeu d'échec doit pouvoir sélectionner, parmi tous les mouvements de pièces et pions possibles, ceux qui sont conformes aux règles du jeu; il doit ensuite "filtrer" l'ensemble des coups possibles de manière à maximiser les chances de victoire. Cette technique de filtration est utile dans des contextes variés. Par exemple, le prédicat `pref_to_tree` (Fig. 11.17) convertit une expression arithmétique préfixée, représentée par sa liste de symboles, en un arbre syntaxique; pour ce faire, il "devine" d'abord comment la liste doit être décomposée en un opérateur et deux opérandes. Le programme ne comporte aucune indication sur la manière de procéder; en fait, le système PROLOG essaie toutes

les décompositions possibles jusqu'à en trouver une adéquate. Cette tactique un peu simpliste est correcte car, si l'expression préfixée d'entrée est bien construite, elle ne peut être décomposée valablement que d'une seule manière. On pourrait objecter qu'il serait plus efficace, plus "intelligent", d'étudier d'abord le problème plus en profondeur, de manière à découvrir un moyen de déterminer la bonne décomposition a priori, sans essais préalables. Un tel moyen existe et peut facilement être programmé mais, pour de nombreux autres problèmes, ce n'est pas le cas; pour ces problèmes, la technique "essais et erreurs" prend tout son sens, ainsi que l'illustrent les exemples développés dans la suite de ce chapitre.[1]

Dans la mesure où la recherche non déterministe dans un espace d'états est une des techniques primordiales de l'intelligence artificielle, il est naturel de penser que le langage PROLOG se prêtera bien à la résolution de problèmes d'intelligence artificielle. Dans ce chapitre, nous allons essayer de montrer que c'est effectivement le cas.

Cette tâche est plutôt malaisée pour plusieurs raisons. D'une part, l'intelligence artificielle est un ensemble hétéroclite de techniques nombreuses, dont certaines n'ont que peu de rapport avec la logique; d'autres sont trop complexes pour être introduites ici; nous nous limiterons donc à ce qui concerne la recherche dans un espace d'états. D'autre part, les applications d'intelligence artificielle typiques sont longues et compliquées; elles nécessitent souvent le recours simultané à plusieurs techniques et se basent généralement sur un domaine de connaissance étendu et pointu.[2] Pour éviter ces écueils, nous choisissons de traiter des problèmes "jouets", dont l'utilité n'est guère apparente. Néanmoins, ces problèmes, par ailleurs amusants, reflètent plusieurs démarches typiques de problèmes plus sérieux et plus réalistes; ils constituent donc une illustration adéquate de notre propos.

12.1. Enigmes attribuables à Lewis Carroll

L'auteur de *Alice in the Wonderland* était un excellent logicien. Il est à l'origine de nombreuses énigmes, puzzles et problèmes logiques, dont certains sont des classiques des tests de quotient intellectuel. Lewis Carroll a eu dans ce domaine — et a toujours — de nombreux émules. Les recueils d'énigmes se sont multipliés lors de ces dernières décennies; de nouvelles sont en permanence

[1] Soulignons aussi le fait que, pour l'utilisateur, le prédicat `pref_to_tree` convient parfaitement. Le fait que l'algorithme sous-jacent ne soit pas "intelligent" a une conséquence positive: le système PROLOG fonctionne (dans ce cas) sans que le programmeur ait eu à lui fournir une tactique intelligente; cela signifie que PROLOG peut pallier l'absence de cette tactique en simulant un comportement intelligent. Nous reviendrons sur ce point au paragraphe 12.1.5.

[2] Un système expert de diagnostic médical, par exemple, comporte notamment une vaste base de données de faits médicaux.

soumises à la sagacité des lecteurs de journaux et de revues, sans qu'il soit généralement possible d'attribuer la paternité de telle énigme à telle personne. Nous pensons que Lewis Carroll n'aurait pas désavoué les quelques problèmes que nous présentons dans ce chapitre, et dont l'origine exacte nous est inconnue.

12.1.1. *Un problème d'intelligence artificielle ?*

Voici l'énoncé d'une énigme classique.

Cinq maisons alignées,
Cinq nationalités,
Cinq couleurs,
Cinq boissons favorites,
Cinq marques de tabac,
Cinq animaux familiers,
et de plus ...

1. *Les numéros des maisons sont 1, 2, 3, 4, 5.*
2. *L'Anglais habite la maison verte.*
3. *L'Espagnol possède un chien.*
4. *On boit du café dans la maison rouge.*
5. *On boit du thé chez l'Ukrainien.*
6. *La maison rouge suit la maison blanche.*
7. *Le fumeur de Old Gold élève des escargots.*
8. *On fume des Gauloises dans la maison jaune.*
9. *On boit du lait au numéro 3.*
10. *Le Norvégien habite au numéro 1.*
11. *Le fumeur de Chesterfield et le propriétaire du renard sont voisins.*
12. *Le fumeur de Gauloises habite à côté du propriétaire du cheval.*
13. *Le fumeur de Lucky Strike boit du jus d'orange.*
14. *Le Japonais fume des Gitanes.*
15. *La maison bleue jouxte celle du Norvégien.*

Qui possède le zèbre ... et qui boit de l'eau ?

On peut discuter la question de savoir si cette énigme est bien un problème d'intelligence artificielle. Cela dépend en fait du point de vue auquel on se place. Chaque maison se décrit par la nationalité de son propriétaire, la couleur de sa façade, un animal familier, une boisson et une marque de tabac. D'un point de vue opérationnel, nous devons sélectionner cinq permutations (indépendantes) de listes de cinq éléments; il existe $(5!)^5 = 120^5 = 24\,883\,200\,000$ manières d'opérer cette sélection. Résoudre ce problème ne requiert en principe guère d'intelligence : il suffit de construire toutes les solutions puis de les tester une à une jusqu'à trouver celle qui vérifie tous les indices qui nous sont donnés. Une meilleure méthode, plus efficace mais moins simple à mettre en œuvre, consiste à construire des solutions partielles et à les rejeter ou les modifier dès que l'un

des indices n'est pas vérifié. C'est en fait la manière dont PROLOG procède, et aussi celle que l'être humain essaiera de mettre en pratique, lors d'un test d'intelligence par exemple. Cela nous incite à considérer qu'il s'agit bien d'un problème d'intelligence artificielle.

12.1.2. *Un programme de résolution en* PROLOG

L'ensemble des définitions auxiliaires codées en PROLOG se trouve à la figure 12.1; on définit les notions d'appartenance, de précédence (une maison précède une autre si le numéro de la première est plus petit que celui de la seconde), de première maison, de maison du milieu et de maisons voisines. Le programme PROLOG permettant de résoudre l'énigme est présenté à la figure 12.2. On observe immédiatement que ce programme n'est rien d'autre qu'un codage de l'énoncé de l'énigme.

```
prc(A,B,[A,B,C,D,E]).  prc(A,C,[A,B,C,D,E]).  prc(A,D,[A,B,C,D,E]).
prc(A,E,[A,B,C,D,E]).  prc(B,C,[A,B,C,D,E]).  prc(B,D,[A,B,C,D,E]).
prc(B,E,[A,B,C,D,E]).  prc(C,D,[A,B,C,D,E]).  prc(C,E,[A,B,C,D,E]).
prc(D,E,[A,B,C,D,E]).

one(A,[A,B,C,D,E]).
three(C,[A,B,C,D,E]).

neighbor(A,B,[A,B,C,D,E]).  neighbor(B,C,[A,B,C,D,E]).
neighbor(C,D,[A,B,C,D,E]).  neighbor(D,E,[A,B,C,D,E]).
neighbor(B,A,[A,B,C,D,E]).  neighbor(C,B,[A,B,C,D,E]).
neighbor(D,C,[A,B,C,D,E]).  neighbor(E,D,[A,B,C,D,E]).

nation(h(N,C,A,B,T),N).
color(h(N,C,A,B,T),C).
animal(h(N,C,A,B,T),A).
drink(h(N,C,A,B,T),B).
tobacco(h(N,C,A,B,T),T).
```

Figure 12.1 *Résoudre l'énigme du Zèbre : les définitions*

La clause principale définit le prédicat go. Les deux premières lignes correspondent à l'indice 1; les lignes suivantes correspondent aux quatorze autres indices, sauf les deux dernières lignes qui correspondent aux deux questions. L'exécution de ce programme est représentée à la figure 12.3.

Cet exemple de l'énigme du zèbre peut être vu comme une parfaite illustration de la puissance de la logique : pour résoudre le problème, il a suffi de le traduire en logique formelle. Le même exemple illustre aussi l'intérêt de PROLOG et, plus généralement, l'intérêt des aspects opératoires de la logique (méthodes de résolution et d'unification). Cet intérêt est spécialement marqué dans le domaine de l'intelligence artificielle, puisque l'utilisateur de PROLOG n'a eu qu'à fournir au système l'énoncé de son problème pour en obtenir la solution.

```
go(X,Y) :- St = [h(N1,C1,A1,B1,T1),h(N2,C2,A2,B2,T2),
                 h(N3,C3,A3,B3,T3),h(N4,C4,A4,B4,T4),h(N5,C5,A5,B5,T5)],
  member(X2,St),nation(X2,english),color(X2,green),
  member(X3,St),nation(X3,spanish),animal(X3,dog),
  member(X4,St),color(X4,red),drink(X4,coffee),
  member(X5,St),nation(X5,ukrainian),drink(X5,tea),
  neighbor(X6a,X6b,St),prc(X6b,X6a,St),color(X6a,red),color(X6b,white),
  member(X7,St),tobacco(X7,oldgold),animal(X7,snails),
  member(X8,St),color(X8,yellow),tobacco(X8,gauloises),
  three(X9,St),drink(X9,milk),
  one(X10,St),nation(X10,norwegian),
  neighbor(X11a,X11b,St),tobacco(X11a,chesterfield),animal(X11b,fox),
  neighbor(X12a,X12b,St),tobacco(X12a,gauloises),animal(X12b,horse),
  member(X13,St),tobacco(X13,luckystrikes),drink(X13,orangejuice),
  member(X14,St),nation(X14,japanese),tobacco(X14,gitanes),
  neighbor(X15a,X15b,St),nation(X15a,norwegian),color(X15b,blue),
  member(Q,St),animal(Q,zebra),nation(Q,X),
  member(R,St),drink(R,water),nation(R,Y).
```

Figure 12.2 *Résoudre l'énigme du Zèbre : le programme*

```
?- go(ZebraOwner,WaterDrinker).
ZebraOwner = japanese
WaterDrinker = norwegian ? ;
no
```

Figure 12.3 *Résoudre l'énigme du Zèbre : la question et les réponses*

Nous avons déjà mentionné que ce problème était "intelligent", puisqu'il est difficile à résoudre par l'être humain; la difficulté vient de l'absence de méthode, et une demi-heure d'essais et d'erreurs, avec crayon et papier, est en général nécessaire pour le résoudre (sans l'aide de PROLOG). On peut donc dire qu'en l'occurrence, la logique formelle et le système PROLOG qui la met en œuvre ont conjointement simulé un comportement intelligent.

Cette argumentation n'est pas fondamentalement incorrecte, mais elle doit être nuancée. L'objection majeure est que, contrairement à ce que l'on pourrait croire, une méthode systématique de résolution de ce type d'énigmes existe : l'être humain peut lui-même simuler PROLOG et résoudre l'énigme en construisant l'arbre de recherche; nous avons vu que cette construction était élémentaire : il suffit de connaître les algorithmes de résolution et d'unification. Cela nous rappelle que le système PROLOG résout cette énigme comme un calculateur électronique évalue des expressions arithmétiques complexes, à savoir en appliquant des méthodes le plus souvent élémentaires, mais à une vitesse inaccessible pour le cerveau humain.

Cette contre-argumentation est elle aussi excessive. Tout d'abord, on ne demande pas à un système informatique d'être intelligent, mais de traiter l'information correctement. Il y a intelligence artificielle dès que les résultats produits par le système requièrent, pour être produit par l'être humain, une démarche intelligente. Le fait est que, pour l'être humain typique, la résolution de l'énigme du zèbre sera vue comme un défi intellectuel, sans qu'il soit question d'appliquer point par point une méthode systématique (ce qui prendrait d'ailleurs un temps très considérable). Le système PROLOG applique une méthode non intelligente (c'est-à-dire, une méthode systématique) mais, ce faisant, il accomplit le même travail que l'être humain appliquant une méthode intelligente (non systématique, faite d'essais et d'erreurs plus ou moins nombreux). Notre conclusion est qu'il y a bien intelligence artificielle, mais l'intelligence artificielle simule l'intelligence naturelle dans ses résultats concrets et non dans les processus calculatoires conduisant à ces résultats.

12.1.3. *La classe de mathématiques*

Cette énigme ressemble à la précédente mais met en évidence quelques limitations de PROLOG.

> *Le professeur de la classe de mathématiques a perdu le classement de ses onze élèves et ne dispose, pour le reconstituer, que des renseignements que veut bien lui donner l'élève Lolo :*
>
> 1. *Le premier et le dernier de la liste portent des noms ayant la même initiale.*
> 2. *Aucun élève n'a la même initiale que son ou ses voisins de classement.*
> 3. *William et Wilfrid sont après Washington qui, au classement, n'est pas voisin de Ludwig.*
> 4. *Herbert, par contre, est au classement, voisin de Ludwig.*
> 5. *Ludovic, voisin de Wolfgang au classement, occupe l'une des trois dernières places.*
> 6. *Le nombre de places séparant Hubert de Ludwig est égal au nombre de places séparant Hubert de William, lequel est à moins de trois places d'Honoré.*
> 7. *Il n'y a pas d'ex-æquo et Wilfrid est avant moi.*
> 8. *Un seul d'entre nous occupe la place qu'il aurait dans l'ordre alphabétique.*
> 9. *Harold occupe une place impaire et est mieux classé que William.*

A priori, ce problème est plus facile que le précédent car le nombre de classements possibles pour onze personnes est "seulement" 11!, c'est-à-dire

39 916 800, alors que l'énigme précédente comportait plus de 24 milliards de possibilités. En outre, le codage des indices reste très simple, puisque ceux-ci ne concernent que les différentes manières de ranger onze éléments dans une liste.

Néanmoins, on observe quelques difficultés inattendues. Le premier type d'indice concerne directement les positions des élèves dans le classement. Deux élèves peuvent être voisins dans la liste (`neighbour`) ou non (`not_neighbour`: au moins un troisième élève se situe entre les deux premiers); un élève peut précéder un autre dans le classement (`before`), occuper une place impaire (`odd_rank`) ou être éloigné de moins de trois places d'un autre (`less_than_three`). Tous ces prédicats s'écrivent sans difficulté; plusieurs programmes sont possibles mais un choix raisonnable est le suivant:

```
neighbour(X,Y,Zs) :- append(Xs,[X,Y|Ys],Zs).
neighbour(Y,X,Zs) :- append(Xs,[X,Y|Ys],Zs).

not_neighbour(X,Y,Zs) :- append(Xs,[A|Ys],Zs),
                         member(X,Xs), member(Y,Ys).
not_neighbour(Y,X,Zs) :- append(Xs,[A|Ys],Zs),
                         member(X,Xs), member(Y,Ys).

before(X,Y,Zs) :- append(Xs,[Y|Ys],Zs), member(X,Xs).

odd_rank(X,[X|Xs]).
odd_rank(X,[A,B|Xs]) :- odd_rank(X,Xs).

less_than_three(X,Y,St) :- append(Xs,[X,Y|Ys],St).
less_than_three(Y,X,St) :- append(Xs,[X,Y|Ys],St).
less_than_three(X,Y,St) :- append(Xs,[X,A,Y|Ys],St).
less_than_three(Y,X,St) :- append(Xs,[X,A,Y|Ys],St).
```

Le prédicat `up_mid` permet de déterminer si trois élèves `A`, `M` et `B` se succèdent dans cet ordre au sein du classement `St`, avec la restriction additionnelle que l'écart entre `A` et `M` est le même que celui entre `M` et `B`. Le prédicat `middle` permet de détecter la situation analogue dans le même ordre ou dans l'ordre inverse.

```
up_mid(A,M,B,St) :- append(Xs,[A,M,B|Ys],St).
up_mid(A,M,B,St) :- append(Xs,[A,A1,M,B1,B|Ys],St).
up_mid(A,M,B,St) :- append(Xs,[A,A1,A2,M,B2,B1,B|Ys],St).
up_mid(A,M,B,St) :- append(Xs,[A,A1,A2,A3,M,B3,B2,B1,B|Ys],St).
up_mid(A,M,B,[A,A1,A2,A3,A4,M,B4,B3,B2,B1,B]).

middle(A,M,B,St) :- up_mid(A,M,B,St).
middle(B,M,A,St) :- up_mid(A,M,B,St).
```

On notera dans ces prédicats l'usage intensif de append. Remarquons aussi que, dans la mesure où la longueur du classement (la structure St) est fixée à 11, on aurait pu programmer odd_rank comme suit :

```
odd_rank(X1,[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11]).
odd_rank(X3,[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11]).
odd_rank(X5,[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11]).
odd_rank(X7,[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11]).
odd_rank(X9,[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11]).
odd_rank(X11,[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11]).
```

C'est peu concis, mais simple et efficace. Notons aussi à cette occasion qu'il n'est pas utile de nommer une variable qui n'intervient qu'une seule fois dans une clause; on aurait pu écrire, par exemple,

```
...
odd_rank(X3,[_,_,X3,_,_,_,_,_,_,_,_]).
...
```

où le caractère "_" est un nom générique pour une variable "anonyme".

D'autres prédicats auxiliaires se réfèrent aux noms des élèves, à l'ordre alphabétique et à l'initiale des noms. On écrit sans peine

```
alpha([harold,herbert,honore,hubert,
       lolo,ludovic,ludwig,
       washington,wilfrid,william,wolfgang]).

init(X,h) :- member(X,[harold,herbert,honore,hubert]).
init(X,l) :- member(X,[lolo,ludovic,ludwig]).
init(X,w) :- member(X,[washington,wilfrid,william,wolfgang]).
```

Ces prédicats permettent de tester qu'un seul élève occupe dans le classement la même place que dans l'ordre alphabétique, et que deux voisins au classement portent toujours des noms dont les initiales sont différentes :

```
exactly_one([X|Xs],[X|Ys]) :- exactly_none(Xs,Ys).
exactly_one([X|Xs],[Y|Ys]) :- X \= Y, exactly_one(Xs,Ys).

exactly_none([],[]).
exactly_none([X|Xs],[Y|Ys]) :- X \= Y, exactly_none(Xs,Ys).

distinct(h,l). distinct(h,w).
distinct(l,h). distinct(l,w).
distinct(w,h). distinct(w,l).

distinct_init_neighbour([X]).
distinct_init_neighbour([X,Y|Ys]) :-
          init(X,I), init(Y,J), I \= J,
```

```
distinct_init_neighbour([Y|Ys]).
```

On peut à présent écrire le prédicat correspondant à la question. Sur base de ce qui précède, on pourra écrire

```
go(St) :-
  St = [X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11],
  init(X1,I), init(X11,I),              /* indice 1  */
  distinct_init_neighbour(St),          /* indice 2  */
  before(washington,william,St),        /* indice 3a */
  before(washington,wilfrid,St),        /* indice 3b */
  not_neighbour(washington,ludwig,St), /* indice 3c */
  neighbour(herbert,ludwig,St),         /* indice 4  */
  neighbour(ludovic,wolfgang,St),       /* indice 5a */
  member(ludovic,[X9,X10,X11]),         /* indice 5b */
  middle(ludwig,hubert,william,St),     /* indice 6a */
  less_than_three(william,honore,St),  /* indice 6b */
  before(wilfrid,lolo,St),              /* indice 7  */
  alpha(L), exactly_one(St,L),          /* indice 8  */
  odd_rank(harold,St),                  /* indice 9a */
  before(harold,william,St).            /* indice 9b */
```

Malheureusement, cette solution a priori correcte ne fonctionne pas en pratique, parce que les indices sont soumis au système dans un ordre inadéquat. En conséquence, lorsque le système PROLOG répond à la question `go(St)`, il génère quantité d'essais infructueux; par exemple, le "classement" :

```
[harold, lolo, harold, ludwig, washington, ludwig,
 william, ludwig, herbert, wilfrid, harold]
```

quoique grossièrement impossible (certains élèves n'apparaissent pas, d'autres apparaissent deux fois ou plus), respecte quand même les quatre premiers indices. Par tâtonnements[3] on crée un ordonnancement plus adéquat pour les indices :

```
go(St) :-
  St = [X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11],
  member(ludovic,[X9,X10,X11]),
  odd_rank(harold,St),
  before(harold,william,St),
  neighbour(ludovic,wolfgang,St),
  neighbour(herbert,ludwig,St),
  before(wilfrid,lolo,St),
  before(washington,william,St),
  not_neighbour(washington,ludwig,St),
```

[3]Et en utilisant quelques règles de bonne pratique non présentées ici.

```
less_than_three(william,honore,St),
before(washington,wilfrid,St),
middle(ludwig,hubert,william,St),
alpha(L), exactly_one(St,L),
init(X1,I), init(X11,I),
distinct_init_neighbour(St).
```

Le système fournit alors l'unique solution au problème; le classement correct est :

```
[washington, herbert, ludwig, wilfrid, harold, lolo,
 hubert, wolfgang, ludovic, honore, william] ;
```

Néanmoins, le temps nécessaire à la résolution du problème est de l'ordre d'une minute sur un ordinateur puissant, alors qu'il n'était que d'une seconde pour le problème précédent. La raison est que, dans l'écriture des prédicats, nous avons privilégié la clarté aux dépens de l'efficacité. Par exemple, le prédicat `not_neighbour` est une source de ralentissement; plutôt que d'en donner une définition explicite, il vaudrait mieux indiquer au système que ce prédicat est vrai si et seulement si le prédicat `neighbour` est faux. PROLOG permet cela; il suffit de remplacer ci-avant la ligne :

```
not_neighbour(washington,ludwig,St),
```

par la ligne :

```
not neighbour(washington,ludwig,St),
```

où le `not` a précisément l'effet souhaité. Ce changement en apparence mineur accélère l'exécution d'un facteur 10 ! Sans entrer dans les détails des aspects "extra-logiques" de PROLOG, signalons que ceux-ci rendent l'emploi du système délicat dans les problèmes complexes; PROLOG n'est donc pas la panacée dans le domaine de la logique appliquée à l'intelligence artificielle, même s'il constitue clairement un des fleurons de ce domaine.

12.1.4. *Les aléas de l'exploration*

Ce nouvel exemple montre comment l'intelligence naturelle et l'intelligence artificielle peuvent utilement collaborer.

> *Un explorateur galactique raconte son arrivée sur une petite planète couverte de forêts et habitée par deux peuples amis, les Véridiques (ils disent toujours la vérité) et les Contradiques (ils ne la disent jamais). Les habitants ne répondent aux questions qu'on leur pose que par "whamm" et "galagala", et n'utilisent jamais un autre moyen d'expression. Certains jours, whamm veut dire "oui" et galagala "non" mais d'autres jours, c'est le contraire ! De plus, un étranger n'a pas le droit de poser plus d'une question à un autochtone.*

Egaré dans la forêt, l'explorateur rencontre un groupe d'habitants au carrefour de deux pistes et veut savoir quelle piste conduit à sa base. Il rapporte le dialogue suivant :

1. *Explorateur : Whamm est-il affirmatif aujourd'hui ?*
   Grocongo : whamm !
2. *Explorateur : Etes-vous tous du même peuple ?*
   Minibola : whamm !
3. *Explorateur : Dodo, Enigmo, Sitopri et toi, êtes-vous du même peuple ?*
   Biribiri : galagala !
4. *Explorateur : Crocro et Minibola sont-ils du même peuple ?*
   Limacela : galagala !
5. *Explorateur : Grocongo, Crocro et toi, êtes-vous du même peuple ?*
   Lasercho : whamm !
6. *Explorateur : Crocro, Limacela et toi, êtes-vous du même peuple ?*
   Albarama : galagala !
7. *Explorateur : Pikaglass et Bonifacio sont-ils du même peuple ?*
   Dada : galagala !
8. *Explorateur : Es-tu du même peuple que Minibola ?*
   Crocro : galagala !
9. *Explorateur : Es-tu du même peuple que Biribiri ?*
   Dodo : whamm !
10. *Explorateur : Sitopri, Dada et toi, êtes-vous du même peuple ?*
    Enigmo : whamm !
11. *Explorateur : Dodo et Enigmo sont-ils du même peuple ?*
    Sitopri : whamm !
12. *Explorateur : Enigmo, Dodo et toi, êtes-vous du même peuple ?*
    Gagagogo : whamm !

A quel peuple appartient Gagagogo ? Quelle question l'explorateur a-t-il posée à Bonifacio, le dernier membre du groupe, pour connaître la direction de sa base ?

On constate qu'à l'exception du premier, tous les indices sont de même nature; ils concernent l'appartenance de certains individus à l'un des deux peuples. La première question est également de ce type, alors que la seconde est de nature différente et semble a priori réclamer un peu de créativité. Ceci suggère de réserver au système PROLOG les onze derniers indices ainsi que la première question, tandis que notre intelligence humaine se chargera du premier indice et de la seconde question.

Il est clair que le premier indice ne permet pas de déterminer si "whamm" veut dire oui ou non, il permet seulement de déterminer que Grocongo est un Véridique. Les onze autres indices requièrent l'écriture d'un prédicat `meme_peuple` vérifiant si tous les éléments d'une liste `L` appartiennent au même peuple. La valeur retournée par ce prédicat dépendra non seulement de `L` mais aussi :

- du peuple auquel appartient le locuteur;
- du fait que "oui" soit "whamm" ou "galagala";
- de la réponse fournie par le locuteur.

Le prédicat à écrire est donc du type `meme_peuple(Ploc,List,Oui,Rep)`. Il y a quatre possibilités, selon que la réponse est affirmative ou négative et que le locuteur est un Véridique ou un Contradique, ce qui conduit immédiatement aux quatre clauses suivantes :

```
meme_peuple(vr,L,Oui,Rep) :- yes(Oui,Rep), meme(L).
meme_peuple(vr,L,Oui,Rep) :- no(Oui,Rep), diff(L).
meme_peuple(ct,L,Oui,Rep) :- yes(Oui,Rep), diff(L).
meme_peuple(ct,L,Oui,Rep) :- no(Oui,Rep), meme(L).
```

Les prédicats auxiliaires sont extrêmement simples :

```
meme([X]).
meme([X,X|Xs]) :- meme([X|Xs]).

some_verid([vr|Xs]).
some_verid([ct|Xs]) :- some_verid(Xs).

some_contra([ct|Xs]).
some_contra([vr|Xs]) :- some_contra(Xs).

diff([vr|Xs]) :- some_contra(Xs).
diff([ct|Xs]) :- some_verid(Xs).

yes(w,w).
yes(g,g).
no(w,g).
no(g,w).
```

Comme précédemment, un prédicat spécial go permettra de répondre à la (première) question. Ce prédicat go prendra comme premier argument `Gag`, qui s'instanciera en le peuple auquel appartient Gagagogo ("`vr`" pour Véridique, "`ct`" pour Contradique). Même si cela n'est pas demandé explicitement, nous essayerons également de savoir quel mot veut dire oui et à quel peuple appartient chacun des protagonistes; cela sera utile pour répondre à la seconde question. On obtient ainsi le prédicat suivant, dans lequel `Struc` (pour

"structure") représente la liste des nationalités des autochtones, dans l'ordre où ils sont évoqués dans l'énoncé:

```
go(Gag,Oui,Struc) :-
  Struc =
   [Gro,Min,Bir,Lim,Las,Alb,Dad,Cro,Dod,Eni,Sit,Gag,Pik,Bon],
  Gro = veri,
  meme_peuple(Min,Struc,Oui,w),
  meme_peuple(Bir,[Dod,Eni,Sit,Bir],Oui,g),
  meme_peuple(Lim,[Cro,Min],Oui,g),
  meme_peuple(Las,[Gro,Cro,Las],Oui,w),
  meme_peuple(Alb,[Cro,Lim,Alb],Oui,g),
  meme_peuple(Dad,[Pik,Bon],Oui,g),
  meme_peuple(Cro,[Min,Cro],Oui,g),
  meme_peuple(Dod,[Bir,Dod],Oui,w),
  meme_peuple(Eni,[Sit,Dad,Eni],Oui,w),
  meme_peuple(Sit,[Dod,Eni],Oui,w),
  meme_peuple(Gag,[Eni,Dod,Gag],Oui,w).
```

Le programme s'exécute comme indiqué à la figure 12.4.[4]

On constate que Gagagogo est nécessairement un Contradique et que, ce jour-là, "whamm" est le mot affirmatif. PROLOG fournit quatre réponses parce que, d'une part, la nationalité d'Alberama n'est pas établie et, d'autre part, Pikaglass et Bonifacio sont de nationalités différentes mais on ne peut préciser qui est le Véridique et qui est le Contradique. Ce dernier point est gênant puisque la seconde question ne peut être résolue aisément que si la nationalité de Bonaficio est connue. Toutefois, il est certain que, de Pikaglass et Bonifacio, un seul est Contradique. L'explorateur peut donc poser à Bonifacio la question suivante: "si je demandais à Pikaglass 'le chemin de droite conduit-il à ma base?', que me répondrait-il?". Si Bonifacio répond "whamm", le chemin à prendre est celui de gauche; s'il répond "galagala", c'est celui de droite. On notera sans surprise que le système PROLOG n'invente pas la question adéquate; il fournit seulement les informations permettant à l'utilisateur de découvrir lui-même cette question.

12.1.5. *Une enquête de Sherlock Holmes*

L'énigme présentée ici illustre un type plus élaboré de collaboration entre l'intelligence naturelle et l'intelligence artificielle; elle permet aussi de montrer le lien étroit existant entre la logique, la programmation logique et la démonstration automatique de théorème.

[4] Pour plus de lisibilité, on a inséré une première ligne spécifiant l'identité des autochtones.

```
%%%%%%%[Gro,Min,Bir,Lim,Las,Alb,Dad,Cro,Dod,Eni,Sit,Gag,Pik,Bon]

?- go(Gag,Oui,Struc).

Gag = ct
Oui = w
Struc = [vr, ct, vr, ct, ct, vr, vr, ct, vr, ct, ct, ct, vr, ct];

Gag = ct
Oui = w
Struc = [vr, ct, vr, ct, ct, vr, vr, ct, vr, ct, ct, ct, ct, vr];

Gag = ct
Oui = w
Struc = [vr, ct, vr, ct, ct, ct, vr, ct, vr, ct, ct, ct, vr, ct];

Gag = ct
Oui = w
Struc = [vr, ct, vr, ct, ct, ct, vr, ct, vr, ct, ct, ct, ct, vr];

No
```

Figure 12.4 *Résolution du problème de l'explorateur*

12.1.6. *L'énoncé*

Un jour, Sherlock Holmes reçoit la visite de son ami Watson que l'on avait chargé d'enquêter sur un assassinat mystérieux datant de plus de dix ans. A l'époque, le duc de Densmore avait été tué par l'explosion d'une bombe, qui avait également détruit le château de Densmore où il s'était retiré. Les journaux d'alors relataient que le testament, détruit lui aussi par l'explosion, avait tout pour déplaire à l'une de ses sept ex-femmes. Or, avant de mourir, le duc les avait toutes invitées à passer quelques jours dans sa retraite écossaise.

H. *Je me souviens de l'affaire. Ce qui est étrange, c'est que la bombe avait été fabriquée spécialement pour être cachée dans l'armure de la chambre à coucher, ce qui montre que l'assassin a nécessairement effectué plus d'une visite au château.*

W. *Certes, et pour cette raison, j'ai interrogé chacune des femmes: Ann, Betty, Charlotte, Edith, Felicia, Georgia, Helen. Elles ont toutes juré qu'elles n'avaient été au château de Densmore qu'une seule fois dans leur vie.*

H. *Hum ! Leur avez-vous demandé à quelle période ont eu lieu leurs séjours respectifs ?*

W. *Hélas ! Aucune ne se rappelait les dates exactes, après plus de dix ans ! Néanmoins, je leur ai demandé qui elles y avaient rencontré :*

- *Ann a rencontré Betty, Charlotte, Felicia et Georgia.*
- *Betty a rencontré Ann, Charlotte, Edith, Felicia et Helen.*
- *Charlotte a rencontré Ann, Betty et Edith.*
- *Edith a rencontré Betty, Charlotte et Felicia.*
- *Félicia a rencontré Ann, Betty, Edith et Helen.*
- *Georgia a rencontré Ann et Helen.*
- *Helen a rencontré Betty, Felicia et Georgia.*

Vous voyez, mon cher Holmes, ces réponses sont concordantes ...

H. *Cela vous étonne ? Ces femmes se méfient ... mais aussi, elles se haïssent ! Aucune n'aurait accepté de s'associer avec une autre, ou de couvrir son crime. Non, il y a une seule coupable ...*

QUI ?

12.1.6.1. *Une solution directe*

Nous savons qui chacune des suspectes a rencontré et aussi que toutes prétendent n'avoir effectué qu'un seul séjour au château. On peut donc essayer, sur base de ces indices, de reconstituer l'ordre des départs et des arrivées dans le temps. Naturellement, comme l'une des suspectes a menti, il ne sera pas possible de mener à bien cette reconstitution, sauf si on ne tient pas compte de la coupable. La démarche consiste donc à déterminer qui doit être éliminée pour qu'un ordonnancement existe, compatible avec le tableau des rencontres. Trois prédicats, `ex_wives`, `same_time` et `with`, modélisent la liste des suspectes et le tableau des rencontres :

```
ex_wives([ann,betty,charlotte,edith,felicia,georgia,helen]).

same_time(ann,[betty,charlotte,felicia,georgia]).
same_time(betty,[ann,charlotte,edith,felicia,helen]).
same_time(charlotte,[ann,betty,edith]).
same_time(edith,[betty,charlotte,felicia]).
same_time(felicia,[ann,betty,edith,helen]).
same_time(georgia,[ann,helen]).
same_time(helen,[betty,felicia,georgia]).

with(A,B) :- same_time(A,L), member(B,L).
```

Le prédicat `history(M,H)` devra produire un ordonnancement des départs et des arrivées `H` compatible avec la partie du tableau des rencontres relative aux suspectes appartenant à la liste `M`. Par exemple, les réponses à la question :

```
?- histoire([ann,betty,edith],H).
```

seront :

```
H = [[ann|in],[betty|in],[ann|out],[edith|in],[betty|out],[edith|out]]
H = [[betty|in],[ann|in],[ann|out],[edith|in],[betty|out],[edith|out]]
H = [[ann|in],[betty|in],[ann|out],[edith|in],[edith|out],[betty|out]]
H = [[betty|in],[ann|in],[ann|out],[edith|in],[edith|out],[betty|out]]
H = [[betty|in],[edith|in],[edith|out],[ann|in],[ann|out],[betty|out]]
H = [[betty|in],[edith|in],[edith|out],[ann|in],[betty|out],[ann|out]]
H = [[edith|in],[betty|in],[edith|out],[ann|in],[ann|out],[betty|out]]
H = [[edith|in],[betty|in],[edith|out],[ann|in],[betty|out],[ann|out]]
```

puisque ces ordonnancements sont compatibles avec les faits : Ann et Betty se sont rencontrées, Betty et Edith se sont rencontrées, Ann et Edith ne se sont pas rencontrées. On construira les ordonnancements par la méthode habituelle : une clause concerne la liste vide, l'autre l'adjonction d'un nouvel élément dans la liste. On observe aussi que, si `A` arrive avant `B`, alors `A` rencontre `B` si et seulement si `B` arrive avant le départ de `A`. On utilise le prédicat `insert_in_order`, tel que la réponse à la question :

```
?- insert_in_order(x,y,L0,L1).
```

où `L0` est une liste et `L1` une variable, soit l'instantiation de `L1` en une liste résultant de l'insertion de `x` et de `y` (dans l'ordre) dans la liste `L1`. Par exemple, l'une des réponses si `L0` est `[a,b,c]` sera `[a,x,b,y,c]`. On obtient aisément les prédicats suivants :

```
history([],[]).
history([A|M],H) :- history(M,H1), h_ins(A,H1,H), ok(A,M,H).

h_ins(A,H1,H) :- insert_in_order([A|in],[A|out],H1,H).

insert_in_order(X,Y,H1,H) :-
     append(Ha,Hx,H1), append(Hb,Hc,Hx),
     append([X|Hb],[Y|Hc],Hbc), append(Ha,Hbc,H).

ok(A,[],H).
ok(A,[B|M],H) :- with(A,B), ok_y(A,B,H), ok(A,M,H).
ok(A,[B|M],H) :- not with(A,B), ok_n(A,B,H), ok(A,M,H).

ok_y(A,B,H) :- before([A|in],[B|in],H), before([B|in],[A|out],H).
ok_y(A,B,H) :- before([B|in],[A|in],H), before([A|in],[B|out],H).

ok_n(A,B,H) :- before([A|out],[B|in],H).
ok_n(A,B,H) :- before([B|out],[A|in],H).

before(X,Y,[X|U]) :- member(Y,U).
```

```
before(X,Y,[_|U]) :- before(X,Y,U).
```

Le prédicat `ok_y(A,B,H)` est vrai si l'ordonnancement `H` est compatible avec le fait que `A` et `B` se sont rencontrés; le prédicat `ok_n(A,B,H)` est vrai si l'ordonnancement `H` est compatible avec le fait que `A` et `B` ne se sont pas rencontrés. La réponse à la question :

```
?- ex_wives(L), history(L,_).
```

est `no` : il n'existe pas d'ordonnancement compatible avec le tableau complet des rencontres. Si on admet l'hypothèse de Sherlock Holmes, il suffit de retirer la coupable du tableau pour qu'un ordonnancement (au moins) soit possible. Cela se fait au moyen des prédicats suivants :

```
guilty(X) :- ex_wives(L), select(X,L,M), history(M,_).

select(X,[X|M],M).
select(X,[Y|M],[Y|N]) :- select(X,M,N).
```

Le prédicat `select` permet d'obtenir une liste `M` en extrayant un élément `X` de la liste `L`. La seule réponse à la question `?- guilty(X)` est `X = ann`.

12.1.6.2. *Une solution plus rapide*

La technique consistant à construire les ordonnancements est simple, naturelle et, bien sûr, correcte, mais elle n'est pas très efficace. On apprend en théorie des graphes et des ordonnancements que l'ordonnancement est impossible s'il existe un "carré contradictoire", c'est-à-dire un groupe de quatre personnes A, B, C et D tel que A et D ont rencontré B et C sans que A ait rencontré D ni que B ait rencontré C. Le prédicat `contra_square` permet de détecter les carrés contradictoires dans le tableau des rencontres :

```
contra_square([A,B,C,D]) :- with(A,B), with(D,B),
                            with(A,C), with(D,C),
                            not with(A,D), not with(B,C).
```

On notera cependant que, si `[A,B,C,D]` est un carré contradictoire, alors `[A,C,B,D]`, `[D,B,C,A]`, `[D,C,B,A]`, `[B,A,D,C]`, `[B,D,A,C]`, `[C,A,D,B]` et `[C,D,A,B]` le sont aussi. Pour éviter ces redondances, on utilisera le prédicat `@<`, correspondant à l'ordre alphabétique. Signalons aussi le prédicat spécial, prédéfini, permettant de regrouper en une liste toutes les solutions à une question donnée; `findall(X,pred(X),Xs)` instancie `Xs` en la liste des solutions à la question `pred(X)`.

```
contra_square([A,B,C,D]) :- with(A,B), A @< B, with(D,B), A @< D,
                            with(A,C), B @< C, with(D,C),
                            not with(A,D), not with(B,C).

contra_list(SQs) :- findall(SQ, contra_square(SQ), SQs).
```

Il reste alors à isoler la coupable, qui intervient dans tous les carrés contradictoires; cela se fait au moyen des prédicats `guilty` et `member_all`:

```
guilty(X) :- contra_list(SQs), member_all(X,SQs).

member_all(X,[]).
member_all(X,[SQ|SQs]):- member(X,SQ), member_all(X,SQs).
```

Cette technique permet de résoudre le problème beaucoup plus rapidement que la précédente. Le temps et les ressources informatiques supplémentaires requises par la première solution représentent le "prix de l'ignorance": le système PROLOG a simulé par une recherche intensive l'absence d'une règle "intelligente" pour résoudre le problème qui lui était soumis. Pour l'utilisateur, les deux solutions se valent dans la mesure où chacune d'elle fournit le résultat correct dans un délai raisonnable. Néanmoins, si la taille du problème avait été plus considérable (par exemple, cinquante suspects au lieu de sept), la première technique aurait été impraticable; cela montre que le recours à l'intelligence artificielle reste un pis-aller, à n'utiliser que si aucune technique plus classique n'est connue.

12.1.6.3. PROLOG *comme outil de preuve*

Il est facile de démontrer au moyen de PROLOG que la présence d'un carré contradictoire empêche la possibilité d'un ordonnancement où chacun des protagonistes n'effectuerait qu'un seul séjour. On peut utiliser le tableau suivant:

```
with(a,d).  with(d,a).  with(b,c).  with(c,b).
```

et d'observer que la question `?- history([a,b,c,d],H)` amène la réponse no.

12.2. Une limitation importante

Les énigmes décrites au début de ce chapitre constituent une application intéressante de la logique, de l'intelligence artificielle et surtout de PROLOG. Le point crucial est le fait qu'il suffit (surtout dans le cas de l'énigme du Zèbre) de donner l'énoncé du problème pour que PROLOG fournisse la solution. Les énigmes suivantes nous ont amenés à nuancer ce succès; ne tenir compte que des aspects logiques, déclaratifs, de PROLOG, en négligeant les aspects opérationnels, conduirait à faire de PROLOG un usage naïf, inefficace et le plus souvent inapproprié. Plus généralement, c'est l'ensemble des techniques de l'intelligence artificielle qu'il convient d'utiliser avec circonspection, et seulement dans le cas où les techniques de programmation plus classiques ne s'appliquent pas. Dans ce paragraphe, nous donnons un (contre-)exemple célèbre, montrant la supériorité de l'approche classique sur l'approche "intelligente". On montre aussi, cependant, que le passage de l'une à l'autre n'est pas une rupture radicale, mais peut se faire méthodiquement.

12.2.1. *Un tri "orienté IA"*

Le problème du tri est l'un des plus étudiés de l'algorithmique classique et donc, a priori, l'un des moins appropriés pour une approche orientée vers l'intelligence artificielle. Il n'est cependant pas difficile, en principe, de résoudre le problème de tri en PROLOG comme s'il s'agissait d'une énigme. En effet, la version triée d'une liste est, par définition, la *permutation ordonnée* de cette liste, ce qui suggère immédiatement le programme suivant :

```
intelsort(Xs,Ys) :- permutation(Xs,Ys), ordered(Ys).
```

Ce programme est acceptable, pour peu que l'on définisse les deux prédicats auxiliaires `permutation` et `ordered`. Ces deux prédicats sont élémentaires et on peut écrire immédiatement

```
permutation([],[]).
permutation([X|Xs],[U|Us]) :-  selection(U,[X|Xs],Zs),
                               permutation(Zs,Us).

selection(X,[X|Xs],Xs).
selection(X,[Y|Ys],[Y|Zs]) :- selection(X,Ys,Zs).

ordered([]).
ordered([X]).
ordered([X,Y|Zs]) :- X =< Y, ordered([Y|Ys]).
```

Cet ensemble de prédicats fonctionne correctement; on a par exemple

```
?- intel_sort([3,2,1,4],Xs).
Xs = [1,2,3,4].
```

Cependant, l'extrême lenteur de l'exécution rend ce programme sans intérêt pratique; il ne permet pas de trier une liste de 20 éléments en un temps raisonnable. Une variante du prédicat `permutation` permet d'améliorer légèrement les choses :

```
permutation([],[]).
permutation([X|Xs],Zs) :- permutation(Xs,Us), selection(X,Zs,Us).
```

Dans la version initiale, le prédicat `selection` était utilisé pour extraire un élément d'une liste; dans la nouvelle version, il est utilisé pour insérer un élément dans une liste. Le problème est toutefois ailleurs; on observe que la méthode de tri consiste à générer des permutations de la liste à trier et à les tester, jusqu'à ce que la bonne permutation soit découverte. Le fait que le nombre $n!$ de permutations augmente très vite en fonction de la longueur n de la liste à trier explique les mauvaises performances de l'algorithme. Plus concrètement, l'approche "intelligente" est ici directement responsable de l'échec. En effet, on demande au système PROLOG de trier une liste sur base d'un "indice" qui permet d'isoler la permutation triée parmi les $n!$ permutations de la liste fournie. Cet indice se révèle insuffisant en pratique.

12.2.2. *Vers un tri raisonnablement efficace*

On peut observer qu'un indice crucial manque à PROLOG: il n'est pas nécessaire de construire entièrement une permutation si la partie déjà construite comporte deux éléments se succédant dans l'ordre décroissant. (On a supposé que le tri se faisait dans l'ordre croissant.) Un premier essai d'intégration de cet indice s'obtient à partir de la première version du prédicat `permutation`:

```
sorted_permutation([],[]).
sorted_permutation([X|Xs],[U|Us]) :-  selection(U,[X|Xs],Zs),
                                      sorted_permutation(Zs,Us),
                                      ordered([U|Us]).
```

Cet essai est clairement infructueux parce que le test `ordered` intervient trop tard. La variante du prédicat `permutation` se prête mieux à l'intégration de l'indice.

```
sorted_permutation([],[]).
sorted_permutation([X|Xs],Zs) :- sorted_permutation(Xs,Us),
                                 selection(X,Zs,Us),
                                 ordered(Zs).
```

Le test `ordered` intervient ici suffisamment tôt pour accélérer radicalement l'exécution du programme. On observe néanmoins que, `Zs` devant être trié, il est indispensable que `Us` le soit aussi, ce qui nous conduit à la variante

```
sorted_permutation([],[]).
sorted_permutation([X|Xs],Zs) :- sorted_permutation(Xs,Us),
                                 ordered(Us),
                                 selection(X,Zs,Us),
                                 ordered(Zs).
```

Le gain d'efficacité est toutefois compromis par un gaspillage; la liste `Zs` résulte de l'insertion dans la liste triée `Us` de l'élément `X`; il ne serait pas nécessaire de tester l'ordonnancement de `Zs` si `X` était inséré à la bonne place. Cela suggère l'amélioration suivante:

```
sorted_permutation([],[]).
sorted_permutation([X|Xs],Zs) :- sorted_permutation(Xs,Us),
                                 ordered(Us),
                                 sorted_select(X,Zs,Us).

sorted_select(X,[X],[]).
sorted_select(X,[X,Y|Ys],[Y|Ys]) :- X =< Y.
sorted_select(X,[Y|Zs],[Y|Ys]) :- X > Y, sorted_select(X,Zs,Ys).
```

Enfin, on montre facilement par récurrence sur la longueur de la liste à trier que, dans le code de `sorted_permutation`, le prédicat `ordered(Us)` est inutile; en effet, chaque insertion se fait maintenant à sa place.

12.2.3. *Approche directe du problème de tri*

On peut aborder le problème du tri de manière entièrement différente. En effet, le tri est une activité banale, à laquelle on se livre fréquemment; il suffira donc de formaliser cette activité pour obtenir un programme de tri. Considérons par exemple le cas d'un joueur de cartes, qui souhaite trier sa main (par exemple dans l'ordre ♡, ♠, ◇, ♣ et, au sein d'une même couleur, dans l'ordre A, R, D, V, 10, 9, 8, 7, 6, 5, 4, 3, 2). Une technique habituelle consiste à ramasser les cartes une à une et à les insérer directement à leur place, au fur et à mesure. Quand le joueur ramasse la treizième carte, il a déjà trié les douze premières; il lui reste à parcourir visuellement son jeu, par exemple de gauche à droite, pour ranger la dernière carte à l'endroit approprié. Cela suggère le code suivant :

```
insert_sort([],[]).
insert_sort([X|Xs],Zs) :- insert_sort(Xs,Us),
                          insert(X,Us,Zs).
```

Insérer une carte dans une main vide est immédiat; si par contre la main comporte au moins une carte, on compare la carte à insérer à la première carte de la main; on obtient aisément le code suivant :

```
insert(X,[],[X]).
insert(X,[Y|Ys],[X,Y|Ys]) :- X =< Y.
insert(X,[Y|Ys],[Y|Zs]) :- X > Y, insert(X,Ys,Zs).
```

On a obtenu un algorithme de tri classique, appelé "tri par insertion". Il est intéressant de noter que, en rebaptisant les prédicats et en inversant les deuxième et troisième arguments du prédicat auxiliaire, on retombe sur la dernière version de l'algorithme de tri développé au paragraphe précédent (prédicat principal `sorted_permutation`, prédicat auxiliaire `sorted_select`).

Une autre technique de tri familière consiste à considérer simultanément tous les objets à trier et à sélectionner successivement le premier, le second (le premier des objets qui restent), etc. Cette idée donne lieu immédiatement au code suivant ("tri par sélection") :

```
sel_sort([],[]).
sel_sort([X|Xs],[Y|Ys]) :- first([X|Xs],Y),
                           selection(Y,[X|Xs],Zs),
                           sel_sort(Zs,Ys).

first([X],X).
first([Y,X|Xs],Z) :- first([X|Xs],U), smallest(Y,U,Z).

smallest(Y,U,Y) :- Y =< U.
smallest(Y,U,U) :- Y > U.
```

12.2.4. *Le tri rapide*

Le "tri des joueurs de cartes", c'est-à-dire le tri par insertion présenté au paragraphe 12.2.3, est qualifié de raisonnablement efficace parce que le temps mis à trier n données est proportionnel au carré de n. En effet, pour trier n données on effectue n insertions dans des listes déjà triées, mais les longueurs respectives de ces listes sont $0, 1, 2, \ldots, n-1$, et le temps nécessaire à réaliser une insertion est naturellement proportionnel à la longueur de la liste dans laquelle on insère.

La méthode dite de tri rapide ("quicksort") procède différemment. Pour trier n objets, on exécute l'algorithme récursif suivant :

- Si $n = 0$ ou $n = 1$, c'est déjà trié;
- Si $n > 1$, on procède comme suit :
  - on choisit un objet, appelé "pivot";
  - on classe les autres objets en "petits" et "grands" par rapport au pivot;
  - les deux parties sont triées séparément ...
  - puis concaténées (sans oublier le pivot) en une liste triée.

Le prédicat de tri rapide se définit immédiatement :

```
quick([], []).
quick([X|Xs], Ys) :- part(X, Xs, Low, High),
                     quick(Low, L), quick(High, H),
                     append(L, [X|H], Ys).
```

Si `part(X, Xs, Low, High)` est vrai, alors `Low` doit contenir les éléments de `Xs` inférieur à `X` tandis que `High` doit contenir les autres. On a

```
part(A, [], [], []).
part(A, [X|Xs], [X|Ys], Zs) :- before(X, A),
                               part(A, Xs, Ys, Zs).
part(A, [X|Xs], Ys, [X|Zs]) :- not before(X, A),
                               part(A, Xs, Ys, Zs).
```

On suppose que le prédicat `before(X,Y)` est vrai si `X` doit précéder `Y` dans le classement, c'est-à-dire si `X` est inférieur à `Y`.

On peut montrer que cet algorithme devient nettement plus efficace que le tri par insertion si le nombre d'objets à trier est relativement élevé et si le pivot est en général bien choisi (les "petits" éléments sont à peu près aussi nombreux que les "grands").

12.2.5. *Le problème des partitions*

L'avantage immédiat du système PROLOG par rapport à d'autres langages de programmation est sa gestion automatique du non-déterminisme. Plus

concrètement, PROLOG peut répondre correctement et exhaustivement aux questions admettant plusieurs réponses. Nous avons vu que cet avantage est décisif dans certains problèmes d'intelligence artificielle mais il est important de souligner son grand intérêt pour des problèmes plus classiques; nous illustrons ce fait en traitant le problème des partitions d'abord classiquement, puis en exploitant le non-déterminisme du prédicat append.

Une partition d'un ensemble E est une famille de parties non vides de E telle que chaque élément de E appartienne à exactement un membre de la famille. Le problème consiste à construire toutes les partitions d'un ensemble donné. A titre d'exemple, les partitions de l'ensemble $\{a, b, c\}$ sont au nombre de cinq:

$\{\{a, b, c\}\}$,
$\{\{a\}, \{b, c\}\}$,
$\{\{a, b\}, \{c\}\}$,
$\{\{a, c\}, \{b\}\}$,
$\{\{a\}, \{b\}, \{c\}\}$.

Ce problème, comme la plupart des problèmes d'analyse combinatoire, se résout par récurrence: comment construit-on une partition d'un ensemble à $n+1$ éléments au départ d'une partition d'un ensemble à n éléments? Il existe deux procédés: on ajoute une partie comportant uniquement le nouvel élément, ou on insère ce nouvel élément dans une partie existante. On résout aussi explicitement le cas de base: un ensemble à 0 élément est l'ensemble vide. Il admet une seule partition, l'ensemble vide lui-même.

L'approche classique pour obtenir les partitions de l'ensemble $\{x\} \cup E$ consiste à construire (en deux exemplaires) la liste des partitions de l'ensemble E puis à transformer ces listes, l'une par le premier procédé, l'autre par le second, pour obtenir les partitions de l'ensemble $\{x\} \cup E$. On représente un ensemble tel que $\{a, b, c\}$ par la liste `[a,b,c]` et une partition telle que $\{\{a, c\}, \{b\}\}$ par la liste `[[a,c],[b]]`. Le programme principal s'obtient immédiatement:

```
c_partitions([],[[]]).
c_partitions([X|Xs],Ps) :- c_partitions(Xs,Qs),
                           proc_1(X,Qs,P1s), proc_2(X,Qs,P2s),
                           append(P1s,P2s,Ps).
```

Appliquer le premier procédé avec l'élément a et la liste des partitions de $\{b, c\}$ consiste à insérer la partie $\{a\}$ dans chacune des partitions, ce qui se réalise facilement:

```
proc_1(X,Qs,Rs) :- insert_in_all([X],Qs,Rs).

insert_in_all(U,[],[]).
insert_in_all(U,[Us|Uss],[[U|Us]|Vss]) :-
              insert_in_all(U,Uss,Vss).
```

Appliquer le second procédé avec l'élément a et la liste des partitions de

$\{b, c\}$ consiste à "éclater" chaque partition en une liste de partitions; par exemple, la partition $\{\{b\}, \{c\}\}$ devient la liste des deux partitions $\{\{a, b\}, \{c\}\}$ et $\{\{b\}, \{a, c\}\}$. Ces listes de partitions sont concaténées en une liste globale. On a :

```
proc_2(X,[],[]).
proc_2(X,[Q|Qs],Ps) :- split(X,Q,R1s), proc_2(X,Qs,R2s),
                       append(R1s,R2s,Ps).

split(X,[],[]).
split(X,[Xs|Xss],[[[X|Xs]|Xss]|Ysss]) :-
     split(X,Xss,Xsss), insert_in_all(Xs,Xsss,Ysss).
```

Tout ceci est raisonnablement simple et on n'a utilisé append que dans le sens direct, pour concaténer des listes. Si maintenant on s'autorise à utiliser la réversibilité de append, on peut apporter au problème des partitions une solution nettement plus simple :

```
partition([],[]).

partition([X|Xs],[[X]|P]) :- partition(Xs,P).

partition([X|Xs],Q) :- partition(Xs,P),
                       append(R1,[R|R2],P),
                       append(R1,[[X|R]|R2],Q).
```

Le prédicat `partition` construit une à une, via son second argument, les partitions de l'ensemble (la liste) spécifié par son premier argument. Soulignons que, dans la dernière clause, la première occurrence de append décompose une liste `P` tandis que la seconde occurrence construit une liste `Q`. On utilise le prédicat spécial `findall` pour regrouper (si on le souhaite) toutes les partitions en une liste :

```
partitions(Xs,Ps) :- findall(P,partition(Xs,P),Ps).
```

("Trouver tous les `P` tels que `partition(Xs,P)` soit vrai et les regrouper dans une liste `Ps`.")

Voici deux exemples d'utilisation :

```
?- c_partitions([a,b,c],Ps).

?- partitions([a,b,c],Ps).
```

La réponse est la même dans les deux cas :

```
?- c_partitions([a,b,c],Ps).
Ps = [[[a], [b], [c]],
      [[a], [b,c]],
      [[a,b], [c]],
      [[b], [a,c]],
      [[a,b,c]]]
```

12.2.6. *Conclusion*

L'expérience confirme qu'un problème pour lequel des mécanismes algorithmiques sont connus n'est pas, ou n'est plus, du ressort de l'intelligence artificielle. Cela n'est pas vraiment étonnant; appliquer une recette met moins de temps, en règle générale, que découvrir cette recette ou apprendre à s'en passer. Par contre, il apparaît que le paradigme de programmation logique est utile tant dans le domaine de l'intelligence artificielle que dans celui de la programmation classique; de plus, la frontière entre ces deux domaines est assez floue.

L'expérimentation menée à propos du problème de tri explique l'influence très importante que peuvent avoir sur la vitesse d'exécution certains "détails" de programmation, comme l'ordre des clauses d'un programme et, plus encore, l'ordre des prédicats au sein d'une clause. L'explication a posteriori et, mieux encore, la prévision de ces variations, ne sont pas du ressort de la logique. Soulignons quand même un principe semblant se dégager de ces expériences; il ne faut pas attendre qu'un objet soit entièrement construit pour le tester et éventuellement le rejeter car cela permet d'épargner le temps consacré à finir la construction d'un objet inutile. Plus simplement, il faut "échouer aussi tôt que possible". Ce principe s'interprète facilement. Résoudre un problème intelligent implique nécessairement le risque de se tromper (si ce risque était absent, le problème ne serait pas qualifié d'intelligent). La clef du succès n'est donc pas d'éviter systématiquement les échecs provisoires mais de détecter ceux-ci au plus tôt, de manière à pouvoir réaliser rapidement d'autres essais.

12.3. Un exemple d'apprentissage

12.3.1. *Introduction*

12.3.1.1. *Hexapion*

Le jeu d'Hexapion introduit au paragraphe 10.2.5 permet d'illustrer simplement une autre potentialité de la programmation logique en intelligence artificielle. On a vu que ce jeu, comme beaucoup de jeux de stratégie, pouvait être décrit, au moins en principe, par un arbre de jeu représentant l'ensemble des parties possibles. Une vue partielle de l'arbre de jeu relatif à l'Hexapion est représenté à la figure 10.7. On conçoit aisément que cet arbre puisse être inclus dans un programme PROLOG, qui permettrait d'obtenir un joueur optimal pour l'Hexapion. Il ne serait pas nécessaire, pour cela, d'avoir développé l'arbre "à la main" : un autre programme PROLOG pourrait construire cet arbre, d'une manière qui le rende directement exploitable par le premier programme.

12.3.1.2. *Principe simple d'apprentissage*

Une autre approche, a priori plus ambitieuse mais aussi plus conforme à l'esprit de l'intelligence artificielle, consiste à construire un programme connaissant les règles du jeu mais pas la stratégie, qu'il doit acquérir en jouant contre un adversaire humain. Il s'agit donc de réaliser un apprentissage artificiel. Quoique la faculté d'apprentissage soit un aspect particulièrement important de l'intelligence, simuler cette faculté est relativement aisé. Il faut cependant observer que le principe même de l'apprentissage suppose que l'entité apprenante (ici, un programme PROLOG) évolue au cours du temps.

Nous supposons que le programme devra jouer systématiquement avec les pions blancs, ce qui, comme indiqué au paragraphe 10.2.5,[5] est un désavantage: si les Noirs ne commettent aucune erreur, les Blancs (qui commencent toujours la partie) perdent toujours. Le principe du programme est qu'il mémorise l'historique des parties qu'il joue, et qu'il ne commette pas deux fois la même erreur. Il peut donc, au bout d'un certain nombre de parties, ne plus jouer du tout, puisqu'une stratégie gagnante existe pour l'adversaire.

A priori, le meilleur choix pour un joueur se trouvant dans cette situation d'infériorité consiste à mettre à profit les erreurs de l'adversaire, qui peut ne pas connaître la stratégie gagnante, et à jouer des coups d'attente, plus ou moins au hasard, quand l'adversaire ne commet pas d'erreur. Dans le cas présent, pour mettre en évidence la faculté d'apprentissage, nous faisons un choix extrême: le programme refuse de jouer et reconnaît sa défaite quand il atteint une situation à partir de laquelle chaque coup possible pour lui s'est déjà avéré perdant lors d'une partie précédente. Le programme considère donc que, tout comme lui, son adversaire apprend et tire parti de ses erreurs passées. Cela signifie que si l'adversaire humain est raisonnablement compétent, le programme finira par refuser de jouer. Dans le cas contraire, il pourra jouer indéfiniment ... ce qui traduit le fait que la qualité de l'apprentissage dépend de la qualité de l'enseignement.

12.3.2. *Le programme de jeu*

12.3.2.1. *Quelques mécanismes particuliers*

Pour permettre l'évolution de sa base de connaissance, le système PROLOG dispose d'instructions spéciales `assert` et `retract` qui lui permettent respectivement d'ajouter à sa base de connaissance une clause (donnée en argument) en cours d'exécution, ou d'en supprimer une.

Le système utilise aussi les instructions `read` et `write`; la première lui permet d'appréhender les coups joués par l'adversaire et la seconde permet

[5] Le lecteur est invité à relire les règles du jeu d'Hexapion dans ce paragraphe.

de communiquer à celui-ci ses propres coups et de récapituler à son intention la situation actuelle du jeu. L'instruction `nl` ("new line") permet d'aller à la ligne.

Un troisième mécanisme, appelé "coupure" et représenté par un point d'exclamation, permet de programmer efficacement certaines situations. Sans entrer dans des détails opératoires qui dépasseraient le cadre de cet exposé introductif, signalons que le fragment de programme:

```
A :- B, !, C.
A :- D.
```

représente souvent:

```
A :- si B alors C sinon D.
```

ce qui dans certains cas pourrait aussi s'écrire:

```
A :- B, C.
A :- not B, D.
```

12.3.2.2. *Le principe du jeu*

Au départ, le programme doit entamer la partie, sauf s'il a déjà déterminé qu'une stratégie gagnante existe pour l'adversaire. Ceci est modélisé par les clauses suivantes:

```
init([p(black, 1, 1), p(black, 2, 1), p(black, 3, 1),
      p(white, 1, 3), p(white, 2, 3), p(white, 3, 3)]).

hexa :- init(S), not bad(S), learn_maybe(S), play(white, S).

hexa :- init(S), bad(S), nl, write('I cannot win.'), nl.
```

Il est avantageux ici d'utiliser la coupure, ce qui donne:

```
hexa :- init(S), not bad(S), !, learn_maybe(S), play(white, S).

hexa :- nl, write('I cannot win.'), nl.
```

La variable S s'instancie en la situation initiale; le prédicat bad s'applique à une situation et est vrai si la situation a été reconnue comme perdante. Le prédicat `learn_maybe` sert à l'apprentissage, nous y reviendrons plus loin. Le prédicat `play` sert à générer les coups. On a:

```
play(C, S) :- chessboard(S), ch_color(C, C1), win(C1, S), !,
              write('The winner is : '), write(C1),
              nl, winner(C1), abort.
play(black, S) :- human(black, S, S_aft), play(white, S_aft).
play(white, S) :- computer(white, S, S_aft), play(black, S_aft).
```

La première clause définissant play correspond à la fin de la partie, les deux autres à un coup du programme et à un coup de l'adversaire humain. Quelques prédicats auxiliaires sont utilisés ici :

```
win(C, S) :- ch_color(C, C1), not legal(C1, M, S).
win(black, S) :- member(p(black, X, 3), S).
win(white, S) :- member(p(white, X, 1), S).

human(C,S1,S2) :- learn_maybe(S1), ask_human(C,M,S1),
                  move(M,S1,S2).

computer(C,S1,S2) :- possibility(C, S1, Ms), !,
                     choose_move(S1, Ms, M), move(M, S1, S2).
computer(C, S1, S2) :- learn(C, S1), end_msg, abort.

winner(black) :- !, maybe(S), forget_maybe(S), learn_bad(S).
winner(white).


end_msg :- nl, write('I know that you are the winner').
```

Le prédicat win permet de détecter la fin de la partie. Le prédicat winner permet au système, en cas de gain de l'adversaire, de faire passer la situation S de la qualification "maybe" (jouable) à la qualification "bad" (perdante). Rappelons que le système refuse de jouer si cela le conduit nécessairement dans une situation perdante.

12.3.2.3. *Les coups possibles*

Le système maintient une liste de coups possibles, qui sont licites et ne conduisent pas à une situation reconnue comme perdante; il donne la priorité à un coup conduisant immédiatement à la victoire, s'il en existe un.

```
choose_move(S, [M|Ms], M1) :- choose_win(S, [M|Ms], M1), !.
choose_move(S, [M|Ms], M).

choose_win(S,[M|Ms],M) :- move(M,S,S1), win(white,S1), !.
choose_win(S,[M|Ms],M1) :- choose_win(S,Ms,M1).

possibility(C, S, [M1|M1s]) :- findall(M, legal(C, M, S), Ms),
        sub_bad(S, Ms, [M1|M1s]), maybe(S1), forget_maybe(S1).

legal(black, m(p(black, X, Y), p(black, X, Y1), forward), S) :-
        member(p(black, X, Y), S), incr(Y, Y1), free(X, Y1, S).
legal(white, m(p(white, X, Y), p(white, X, Y1), forward), S) :-
        member(p(white, X, Y), S), decr(Y, Y1), free(X, Y1, S).
```

```
legal(C, m(p(C, X, Y), p(C, X1, Y1), eat), S) :-
        member(p(C, X, Y), S), diag(C, X, Y, X1, Y1),
        ch_color(C, C1), member(p(C1, X1, Y1), S).

move(m(P1, P2, forward), S_bef, S_aft) :-
     sub(P1, P2, S_bef, S), sort(S, S_aft).
move(m(P1, P2, eat), S_bef, S_aft) :- P2 = p(C2, X, Y),
     ch_color(C2, C), remove(p(C, X, Y), S_bef, S1),
     sub(P1, P2, S1, S), sort(S, S_aft).
```

Les mouvements effectués impliquent une gestion de la configuration des pions sur l'échiquier; nous ne rentrons pas ici dans les détails. Le prédicat `sort` permet de trier la configuration, vue comme la liste des positions des pions. Nous donnons sans commentaires les clauses servant à manipuler les configurations:

```
is_range(X)  :- X>0, X=<3.

incr(X, X1) :- X<3, X1 is X+1.
decr(X, X1) :- X>1, X1 is X-1.

diag(black, X, Y, X1, Y1) :- incr(Y, Y1), incr(X, X1).
diag(black, X, Y, X1, Y1) :- incr(Y, Y1), decr(X, X1).
diag(white, X, Y, X1, Y1) :- decr(Y, Y1), incr(X, X1).
diag(white, X, Y, X1, Y1) :- decr(Y, Y1), decr(X, X1).

ch_color(white, black).        ch_color(black, white).

free(X, Y, S) :- not member(p(C,X,Y), S).
```

12.3.3. *Le dialogue avec l'adversaire*

Nous donnons aussi sans commentaire les clauses qui régissent le dialogue avec l'adversaire humain:

```
ask_human(C, m(P1,P2,Ty), S) :- can_move_it(C, P1, S),
                                where(m(P1,P2,Ty), S).

read_x(X) :- write('X: '), read(X), is_range(X), !.
read_x(X) :- read_x(X).

read_y(Y) :- write('Y: '), read(Y), is_range(Y), !.
read_y(Y) :- read_y(Y).

coord(X, Y) :- write('Enter the coordinate: '),
               nl, read_x(X), read_y(Y).
```

```
which_pawn(C, p(C, X, Y), S) :- nl, write('Which '), write(C),
   write(' pawn do you want to move ?'),
   nl, coord(X, Y), member(p(C, X, Y), S), !.
which_pawn(C, P, S) :-
   write('There is not a '), write(C), write(' pawn there !'),
   nl, which_pawn(C, P, S).

can_move_it(C, P, S) :-
   which_pawn(C, P, S), legal(C, m(P, P1, Type), S), !.
can_move_it(C, P, S) :-
   write('There is not a legal move for it!'),
    nl, can_move_it(C, P, S).

where(m(p(C, X1, Y1), p(C, X2, Y2), Type), S) :- nl,
   write('Where do you want to move it ?'), nl, coord(X2, Y2),
   legal(C, m(p(C, X1, Y1), p(C, X2, Y2), Type), S), !.
where(M, S) :- write('This is not a legal move !'),
               nl, where(M, S).
```

12.3.3.1. *Gestion de l'apprentissage*

Le prédicat show permet l'affichage, à tout moment de la partie, des situations considérées comme mauvaises par le système. Les prédicats learn et forget permettent d'actualiser les listes de situations reconnues comme jouables ou comme mauvaises. On a:

```
show :- findall(S, bad(S), Ss), show_sit(Ss).

learn(C, S) :- findall(M, legal(C, M, S), Ms),
     forget(S, Ms), maybe(S1), forget_maybe(S1), learn_bad(S1).

forget(S, [M|Ms]) :- move(M, S, S1),
                     forget_bad(S1), forget(S, Ms).
forget(S, []).

sub_bad(S1, [M|M1s], M2s) :- move(M, S1, S2), bad(S2), !,
                             sub_bad(S1, M1s, M2s).
sub_bad(S1, [M|M1s], [M|M2s]) :- sub_bad(S1, M1s, M2s).
sub_bad(S1, [], []).

show_sit([[P|Ps]|Ss]) :- chessboard([P|Ps]), show_sit(Ss).
show_sit([[]|Ss]) :- show_sit(Ss).
show_sit([]).
```

```
learn_bad([P|Ps]) :- asserta((bad([P|Ps]))).
learn_bad([]).

forget_bad([P|Ps]) :- retract((bad([P|Ps]))).
forget_bad([]).

learn_maybe([P|Ps]) :- asserta((maybe([P|Ps]))).
learn_maybe([]).

forget_maybe([P|Ps]) :- retract((maybe([P|Ps]))).
forget_maybe([]).

bad([]).

maybe([]).
```

12.3.3.2. *Prédicats auxiliaires supplémentaires*

Nous donnons ici la liste des prédicats auxiliaires permettant notamment de dessiner l'échiquier à l'écran et de trier une situation.

```
chessboard(S) :- nl, nl, draw_coord_col, draw_border,
                 draw_board(1, S), draw_border, nl.

draw_board(Y,S) :- draw_line(Y,S), incr(Y,Y1), !,
                   draw_board(Y1,S).
draw_board(Y,S).

draw_line(Y,S) :- write(' '),write(Y),write(' |'),
                  draw_line(1,Y,S),write('|'),nl.
draw_line(X,Y,S) :- draw_cell(X,Y,S),incr(X,X1),!,
                    draw_line(X1,Y,S).
draw_line(X, Y, S).

draw_cell(X, Y, S) :- member(p(C, X, Y), S), !, draw_pawn(C).
draw_cell(X, Y, S) :- write(' ').

draw_pawn(white) :- write('W').
draw_pawn(black) :- write('B').

draw_coord_col :- Xd is 3//10, write('    '), draw_ten(1, Xd),
                  nl, write('    '), draw_num(3), nl.

draw_ten(N,M) :- N=<M,!,write('         '),write(N),
```

```
                N1 is N+1, draw_ten(N1,M).
draw_ten(N, M).

draw_num(N) :- N>9,number(9,Xs),draw(Xs),write('0'),
               N1 is N-10, draw_num(N1).
draw_num(N) :- N=<9, N>0, number(N, Xs), draw(Xs).
draw_num(0).

draw_border :- make_list(3,'-',Bord), write('   +'),
               draw(Bord), write('+'), nl.

remove(X, [X|Xs], Xs).
remove(X, [Y|Ys], [Y|Zs]) :- X\==Y, remove(X, Ys, Zs).

sub(X, Y, [X|Xs], [Y|Xs]).
sub(X, Y, [Z|Xs], [Z|Ys]) :- sub(X, Y, Xs, Ys).

make_list(N, X, [X|Xs]):- N>0, N1 is N-1, make_list(N1, X, Xs).
make_list(0, X, []).

draw([X|Xs]) :- write(X), draw(Xs).
draw([]).

number(N, Xs) :- number(1, N, Xs).
number(N, N, [N]).
number(S, N, [S|Xs]) :- S<N, S1 is S+1, number(S1, N, Xs).

before(p(black, X1, Y1), p(white, X2, Y2)) :- !.
before(p(C, X1, Y1), p(C, X2, Y2)) :- X1 < X2, !.
before(p(C, X1, Y1), p(C, X2, Y2)) :- Y1 < Y2.

sort  /* procedure de tri, voir section 3.2 */
```

12.3.4. *Exemples d'exécution*

Nous avons donné le texte complet du programme de jeu mais le lecteur non informaticien préférera appréhender les principes de ce programme par le biais de quelques exécutions commentées.

12.3.4.1. *Première partie*

Le système joue sans tactique mais en respectant les règles. Il joue avec les pions `W` (white); l'adversaire humain répond avec les pions `B` (black).

```
?- hexa.

    123      123
   +---+    +---+
 1 |BBB| 1  |BBB|
 2 |   | 2  |W  |
 3 |WWW| 3  | WW|
   +---+    +---+

Which black pawn do you want to move ?
Enter the coordinate:
X: 2.
Y: 1.

Where do you want to move it ?
Enter the coordinate:
X: 1.
Y: 2.

    123      123
   +---+    +---+
 1 |B B| 1  |B B|
 2 |B  | 2  |BW |
 3 | WW| 3  |  W|
   +---+    +---+
```

Le haut de cette figure représente la situation initiale et la situation après le premier coup du système. Le milieu représente le dialogue avec l'adversaire humain, qui déclare vouloir prendre le pion blanc avancé au moyen du pion noir médian; celui-ci passe donc de sa position initiale `(2,1)` à la position `(1,2)`, comme indiqué en bas à gauche de la figure. Le système réagit en avançant son propre pion médian, ce qui conduit à la situation représentée en bas à droite de la figure.

En présence d'un adversaire humain jouant convenablement, la partie s'achève immédiatement :

```
    123      123
   +---+    +---+
 1 |B B| 1  |B B|
 2 |BW | 2  | W |
 3 |  W| 3  |B W|
   +---+    +---+

The winner is : black
Execution Aborted
```

Les Noirs ont gagné parce qu'un de leur pion a atteint la ligne opposée. Le système apprend à chaque défaite; il mémorise le fait que sa dernière position était perdante, ce que l'on met en évidence en appelant le prédicat sans argument `show`:

```
?- show.

    123
   +---+
 1 |B B|
 2 |BW |
 3 |  W|
   +---+
```

12.3.4.2. *Deuxième partie*

Nous donnons simplement la liste des situations successives. Rappelons que le système joue toujours le premier, avec les Blancs (`W`).

```
   123      123      123      123      123
  +---+    +---+    +---+    +---+    +---+
1 |BBB| 1 |BBB| 1 |B B| 1 |B B| 1 |B B|
2 |   | 2 |W  | 2 |B  | 2 |B W| 2 |  W|
3 |WWW| 3 | WW| 3 | WW| 3 | W | 3 |BW |
  +---+    +---+    +---+    +---+    +---+

The winner is : black
Execution Aborted
```

Le système a de nouveau perdu, il mémorise une nouvelle situation perdante:

```
   123      123
  +---+    +---+
1 |B B| 1 |B B|
2 |B W| 2 |BW |
3 | W | 3 |  W|
  +---+    +---+
```

12.3.4.3. *Progression de l'apprentissage*

Supposons que l'adversaire humain commette une erreur; le système peut éventuellement en profiter et gagner la partie mais, dans ce cas, il n'apprend rien et sa liste de mauvaises situations reste inchangée.

A l'issue d'une troisième partie perdante, la liste des mauvaises situations sera:

```
   123      123      123
  +---+    +---+    +---+
1 |B B| 1  |B B| 1  |B B|
2 |W  | 2  |B W| 2  |BW |
3 |  W| 3  | W | 3  |  W|
  +---+    +---+    +---+
```

Si l'adversaire humain ne commet pas d'erreur, la partie suivante pourrait être celle-ci :

```
   123      123      123
  +---+    +---+    +---+
1 |BBB| 1  |BBB| 1  |B B|
2 |   | 2  |W  | 2  |B  |
3 |WWW| 3  | WW| 3  | WW|
  +---+    +---+    +---+
```

```
I know that you are the winner
```

Le système renonce à continuer la partie parce qu'il n'a que trois possibilités, chacune d'elle conduisant à une situation interdite :

- Prendre le pion noir avancé avec le pion blanc médian.
- Avancer le pion blanc latéral.
- Avancer le pion blanc médian.

Cette nouvelle défaite du système améliore l'apprentissage; le système sait maintenant que la situation résultant d'une entame latérale est perdante, ce que l'on met en évidence en invoquant le prédicat `show` :

```
?- show.

   123
  +---+
1 |BBB|
2 |W  |
3 | WW|
  +---+
```

Lors de la partie suivante, le système effectuera une entame centrale :

```
?- hexa.

   123      123      123      123      123
  +---+    +---+    +---+    +---+    +---+
1 |BBB| 1  |BBB| 1  | BB| 1  | BB| 1  | BB|
2 |   | 2  | W | 2  | B | 2  |WB | 2  |W  |
3 |WWW| 3  |W W| 3  |W W| 3  |  W| 3  | BW|
  +---+    +---+    +---+    +---+    +---+
```

```
The winner is : black
Execution Aborted
```

Après avoir perdu une vingtaine de parties, et ceci indépendamment du nombre de parties qu'il peut avoir gagnées, l'ensemble des mauvaises situations est le suivant :

```
   123      123      123
  +---+    +---+    +---+
1 |BBB| 1 |BBB| 1 |BBB|
2 |W  | 2 | W | 2 |  W|
3 | WW| 3 |W W| 3 |WW |
  +---+    +---+    +---+
```

Il n'y a donc plus d'entame jouable; la partie suivante sera :

```
   123
  +---+
1 |BBB|
2 |   |
3 |WWW|
  +---+
```

```
I know that you are the winner
Execution Aborted
```

Le système renonce immédiatement; il apprend de ce fait qu'il a perdu avant de commencer, du moins si l'adversaire humain ne commet pas d'erreur; il a donc appris qu'il existait une stratégie gagnante pour son adversaire :

```
   123
  +---+
1 |BBB|
2 |   |
3 |WWW|
  +---+
```

```
I cannot win.
```

12.3.4.4. *Conclusion*

Le programme que nous avons présenté ici met bien en évidence la manière dont l'apprentissage peut s'intégrer à la programmation logique. On pourrait sans difficulté perfectionner le programme pour qu'il tienne indifféremment le rôle des Blancs ou celui des Noirs, et aussi pour que, en situation perdante, il joue un coup d'attente plutôt que de renoncer. Le système se comporterait alors comme un joueur en progrès, nul au début, puis graduellement optimal au fur et à mesure qu'il tire les leçons des parties perdues.

Il ne faudrait évidemment pas en déduire que quelques lignes de programme suffisent à doter un système d'une capacité d'apprentissage importante. En effet, si l'approche présentée ici convient parfaitement à un jeu tel que Hexapion, c'est parce qu'une partie d'Hexapion ne peut durer que six coups; le nombre de parties distinctes est donc relativement limité. Sur le plan théorique, la même technique pourrait être utilisée pour l'apprentissage des Echecs mais, une partie d'Echecs pouvant comporter une centaine de coups, l'acquisition d'une stratégie gagnante (s'il en existe une) prendrait un temps énorme; quels que soient les progrès futurs de la technologie des ordinateurs, il est impensable d'obtenir un programme de jeu performant par une technique d'apprentissage aussi simple.

12.4. Pour en savoir plus

Nous avons déjà mentionné à la fin du chapitre précédent diverses références concernant PROLOG et l'intelligence artificielle. La rubrique tenue par Martin Gardner dans la revue *Scientific American* a influencé le choix des applications que nous avons traitées ici. Tant pour les énigmes que pour l'intelligence artificielle elle-même, nous renvoyons le lecteur aux nombreux livres et recueils regroupant ces rubriques [Gardner]. C'est dans l'une d'elles, dix ans avant l'invention de PROLOG, qu'a été introduit le programme d'apprentissage d'Hexapion développé ici.

Chapitre 13
Vérification assistée par ordinateur

13.1. Introduction

L'une des plus récentes et prometteuses applications de la logique est la vérification formelle des programmes. Dans le volume précédent, nous avons présenté la méthode des invariants, tant dans le cadre de la programmation séquentielle (§ II.3) que dans celui de la programmation parallèle (§ II.4). En pratique, c'est surtout dans le domaine des programmes parallèles que la méthode des invariants est utilisée, parce que ces programmes sont difficiles à mettre au point, ce qui justifie l'investissement relativement lourd induit par la vérification formelle.

Rappelons d'abord brièvement ce qu'est un programme et ce qu'est un invariant.[1] Dans le présent contexte, un programme est un texte formel qui spécifie des trajectoires dans un certain espace, appelé *espace des états* ou *espace d'état*. Cet espace est le produit cartésien d'une famille d'ensembles, dont chaque membre correspond à l'ensemble des valeurs que peut prendre une *variable* du programme. Lors de l'exécution du programme, chaque instruction (ou transition) exécutée induit le passage d'un point de l'espace d'état à un autre. La plupart des propriétés de programme peuvent s'écrire sous la forme suivante :

> Si l'exécution du programme $\mathcal{S}$ commence dans un état $s_0 \in A$,
> alors, tout état ultérieur reste dans l'ensemble J.

[1] Pour une présentation détaillée, nous renvoyons le lecteur aux chapitres 3 et 4 du volume précédent.

Ce type de propriété s'appelle propriété d'invariance ou propriété de sûreté.

Les sous-ensembles A et J de l'espace d'état sont généralement décrits par des formules de logique appelées assertions, notées également A et J.[2] Pour vérifier la propriété ci-dessus, on essaie de construire une assertion I, vérifiant trois contraintes :

1. $\models A \Rightarrow I$;
2. $\models I \Rightarrow J$;
3. pour toute transition $\tau \in \mathcal{S}$, $\{I\}\,\tau\,\{I\}$.

Une assertion vérifiant la dernière contrainte est un *invariant* du programme $\mathcal{S}$. Rappelons que le triplet de Hoare $\{A\}\tau\{B\}$ signifie que l'exécution de la transition τ au départ d'un état vérifiant A conduit à un état vérifiant B.

Le bien-fondé de la méthode des invariants est évident. Si I vérifie les trois contraintes, alors l'état initial de toute exécution vérifie A et aussi I (première contrainte). L'état successeur de tout état vérifiant I vérifie aussi I (troisième contrainte) donc tout état de l'exécution vérifie I et aussi J (deuxième contrainte). Réciproquement, nous verrons au paragraphe 13.2.1 que si un programme est correct par rapport à une propriété de sûreté, ce fait peut toujours être établi, du moins en principe, par la méthode des invariants.

Nous illustrons tous ces concepts au moyen d'une version élémentaire de l'algorithme d'Euclide permettant de calculer le plus grand commun diviseur de deux entiers strictement positifs. On décrit cet algorithme par le triplet $(P, \mathcal{M}, \mathcal{T})$ où

$$P = \{\ell_0, \ell_1, \ell_2, \ell_3, \ell_4\}\,;$$

$$\mathcal{M} = \{x, y, g \ : \text{entiers strictement positifs}\}\,;$$

$$\begin{aligned}
\mathcal{T} = \{\,&\alpha : \ (\ell_0,\ x > y \longrightarrow \text{skip},\ \ell_1)\,,\\
&\beta : \ (\ell_0,\ x < y \longrightarrow \text{skip},\ \ell_2)\,,\\
&\gamma : \ (\ell_0,\ x = y \longrightarrow \text{skip},\ \ell_3)\,,\\
&\delta : \ (\ell_1,\ x := x - y,\ \ell_0)\,,\\
&\varepsilon : \ (\ell_2,\ y := y - x,\ \ell_0)\,,\\
&\zeta : \ (\ell_3,\ g := x,\ \ell_4)\,\}\,.
\end{aligned}$$

[2] A l'inclusion ensembliste $A_1 \subset A_2$ correspond la notion de conséquence logique $A_1 \models A_2$ ou $\models A_1 \Rightarrow A_2$; notons qu'une assertion A_1 est plus forte qu'une assertion A_2 si l'ensemble correspondant A_1 est inclus dans l'ensemble A_2. "Plus fort" signifie ici "plus petit"; c'est naturel dans la mesure où une assertion plus forte est moins souvent vraie qu'une assertion plus faible.

Les points de contrôle forment le processus P et les variables x, y et g forment la mémoire du système. L'espace d'état est donc le produit cartésien $\Sigma =_{def} P \times \mathbb{N}^3$. Le calcul du plus grand commun diviseur des nombres 16 et 10 donne lieu à l'exécution $(s_0, \ldots, s_9)$; chaque état $s_i \in \Sigma$ est représenté par une assertion spécifiant le point de contrôle et les valeurs des trois variables (la valeur spéciale $\perp$ représente une valeur inconnue, quelconque ou non définie). On peut décrire l'exécution par le tableau suivant, qui spécifie les états successifs (à droite) et les transitions entre ces états (à gauche):

$$
\begin{array}{cl}
 & s_0 : (at\ \ell_0 \wedge x = 16 \wedge y = 10 \wedge g = \perp) \\
\alpha & \\
 & s_1 : (at\ \ell_1 \wedge x = 16 \wedge y = 10 \wedge g = \perp) \\
\delta & \\
 & s_2 : (at\ \ell_0 \wedge x = 6 \wedge y = 10 \wedge g = \perp) \\
\beta & \\
 & s_3 : (at\ \ell_2 \wedge x = 6 \wedge y = 10 \wedge g = \perp) \\
\varepsilon & \\
 & s_4 : (at\ \ell_0 \wedge x = 6 \wedge y = 4 \wedge g = \perp) \\
\alpha & \\
 & s_5 : (at\ \ell_1 \wedge x = 6 \wedge y = 4 \wedge g = \perp) \\
\delta & \\
 & s_6 : (at\ \ell_0 \wedge x = 2 \wedge y = 4 \wedge g = \perp) \\
\beta & \\
 & s_7 : (at\ \ell_2 \wedge x = 2 \wedge y = 4 \wedge g = \perp) \\
\varepsilon & \\
 & s_8 : (at\ \ell_0 \wedge x = 2 \wedge y = 2 \wedge g = \perp) \\
\zeta & \\
 & s_9 : (at\ \ell_4 \wedge x = 2 \wedge y = 2 \wedge g = 2)
\end{array}
$$

Remarque. Un processus induit toujours une variable spéciale, appelée point de contrôle ou compteur de programme. Dans le cas présent, cette variable peut prendre les valeurs $\ell_0, \ell_1, \ell_2, \ell_3, \ell_4$. Comme dans le volume II, nous écrirons, par exemple, $at\ \ell_2$ au lieu de $\ell = \ell_2$. Les expressions $at\ \ell_i$ sont appelées prédicats de place. Dans tout état, accessible ou non, de l'espace d'état Σ, un et un seul prédicat de place est vrai; dans le cadre d'un programme parallèle, cette règle est valable pour chaque processus pris individuellement.

La propriété intéressante de l'algorithme de calcul du plus grand commun diviseur donné plus haut s'exprime simplement:

> *Si l'exécution du programme commence dans un état vérifiant l'assertion* $A =_{def} (at\ \ell_0 \wedge x = a \wedge y = b)$, *alors tout état ultérieur vérifie l'assertion* $J =_{def} (at\ \ell_4 \Rightarrow g = \gcd(a, b))$.

Dans cet énoncé, a et b désignent deux entiers strictement positifs et $\gcd(a, b)$ désigne leur plus grand commun diviseur.

Pour démontrer cette propriété, on utilise l'invariant

$$
\begin{aligned}
I =_{def} \quad & (at\ \ell_0 \Rightarrow [0 < x \leq a \wedge 0 < y \leq b \wedge \gcd(x, y) = \gcd(a, b)]) \\
\wedge \quad & (at\ \ell_1 \Rightarrow [0 < x \leq a \wedge 0 < y \leq b \wedge x > y \wedge \gcd(x, y) = \gcd(a, b)]) \\
\wedge \quad & (at\ \ell_2 \Rightarrow [0 < x \leq a \wedge 0 < y \leq b \wedge x < y \wedge \gcd(x, y) = \gcd(a, b)]) \\
\wedge \quad & (at\ \ell_3 \Rightarrow [0 < x \leq a \wedge 0 < y \leq b \wedge x = y = \gcd(a, b)]) \\
\wedge \quad & (at\ \ell_4 \Rightarrow [0 < x \leq a \wedge 0 < y \leq b \wedge x = y = g = \gcd(a, b)])\,.
\end{aligned}
$$

Cette technique est conceptuellement simple mais demande un travail assez lourd. Dans l'exemple présent, il est immédiat d'observer que les deux premières contraintes relatives à l'assertion I sont vérifiées. Pour vérifier la troisième, c'est-à-dire pour montrer que I est un invariant, il faut montrer que I est respecté par chaque transition, c'est-à-dire que tous les triplets

$$\{I\}\,\alpha\,\{I\},\ldots,\{I\}\,\zeta\,\{I\}$$

sont vérifiés. A titre d'exemple, considérons le cas de la transition δ. Avec les notations du paragraphe II.3.4.3, le triplet

$$\{I\}\,\delta\,\{I\}$$

peut se récrire[3]

$$\{I[at\ \ell_1]\}\ x := x - y\ \{I[at\ \ell_2]\}$$

ou encore

$$\{0 < x \leq a \wedge 0 < y \leq b \wedge x > y \wedge \gcd(x,y) = \gcd(a,b)\}$$
$$x := x - y$$
$$\{0 < x \leq a \wedge 0 < y \leq b \wedge \gcd(x,y) = \gcd(a,b)\}$$

On peut réduire ce triplet à une formule de logique classique en utilisant l'axiome de Hoare (cf. § II.3.5.4):

$$\{A(e_1,e_2)\}\ (x,y) := (e_1,e_2)\ \{A(x,y)\}$$

Cela nous permet de réduire la validité du triplet à la validité de la formule

$$[0 < x \leq a \wedge 0 < y \leq b \wedge x > y \wedge \gcd(x,y) = \gcd(a,b)]$$
$$\Rightarrow$$
$$[0 < x - y \leq a \wedge 0 < y \leq b \wedge \gcd(x-y,y) = \gcd(a,b)]$$

Cette implication découle de propriétés arithmétiques évidentes, telles que

$$(0 < y \wedge x > y) \Rightarrow 0 < x - y$$

et

$$(x \leq a \wedge 0 < y) \Rightarrow x - y \leq a$$

et aussi d'un théorème arithmétique élémentaire attribué à Euclide:

$$x > y > 0 \Rightarrow \gcd(x-y,y) = \gcd(x,y)\,;$$

On utilise aussi des règles purement logiques, comme

$$(x = z \wedge y = z) \ \Rightarrow\ x = y\,.$$

---

[3]Rappelons ici un point de terminologie. La notation $A[at\ \ell]$ (lire "A au point ℓ") représente l'assertion A quand le prédicat de place $at\ \ell$ est vrai. Formellement, $A[at\ \ell]$ s'obtient à partir de A en remplaçant toutes les occurrences de $at\ \ell$ par vrai et toutes les occurrences de $at\ \ell'$ par faux, où ℓ' est n'importe quel point de contrôle distinct de ℓ mais appartenant au même processus que ℓ. Dans le cas présent, il y a un seul processus donc, par exemple, $I[at\ \ell_1]$ s'obtient en remplaçant, dans I, $at\ \ell_1$ par vrai et $at\ \ell_0$, $at\ \ell_2$, $at\ \ell_3$ et $at\ \ell_4$ par faux.

Cela n'est pas difficile mais devient long et fastidieux même pour des programmes de taille et de complexité modérées. En fait, on a autant de chances (au moins) de se tromper en vérifiant un programme qu'en écrivant ce programme et ses propriétés. Un autre désavantage de la méthode des invariants est que la première étape est la construction de l'assertion I; même pour les programmes simples, la découverte d'une assertion susceptible d'être un invariant approprié n'est pas évidente.

Ces inconvénients majeurs de la méthode des invariants pourraient être éliminés si l'on disposait, d'une part, d'un système permettant la découverte de "candidats invariants" et, d'autre part, d'un système permettant de vérifier qu'un candidat invariant est — ou n'est pas — un invariant approprié (c'est-à-dire vérifiant aussi les deux premières contraintes). Ces deux problèmes sont du ressort de l'intelligence artificielle et constituent des applications intéressantes de la logique. Nous les abordons brièvement dans la suite de ce chapitre.

13.2. Construction d'invariant

13.2.1. *Une méthode théorique*

Les données du problème sont un programme $\mathcal{S}$ (séquentiel ou parallèle), une condition initiale A et une propriété de sûreté J. Les assertions A et J représentent respectivement l'ensemble des états de l'espace $\Sigma_{\mathcal{S}}$ admissibles comme état initial d'une exécution et comme état quelconque d'une exécution. Le premier ensemble doit nécessairement être inclus dans le second, ce qui, en termes d'assertions, signifie que J est conséquence logique de A.

Le programme $\mathcal{S}$ permet de déterminer, pour tout état $s \in \Sigma$, l'ensemble $post(s)$ des états successeurs, qui peuvent suivre immédiatement s dans une exécution, et de même l'ensemble $pre(s)$ des états prédécesseurs, qui peuvent précéder immédiatement s dans une exécution. Observons que, dans le cas d'un programme déterministe, l'ensemble $post(s)$ comporte au plus un élément. Si E est un sous-ensemble de Σ, on note naturellement $post(E)$ et $pre(E)$ l'ensemble des successeurs des éléments de E et l'ensemble des prédécesseurs des éléments de E, respectivement; on note $\widetilde{pre}(E)$ l'ensemble des états dont tous les successeurs appartiennent à E.[4]

Si un invariant adéquat[5] I existe, l'ensemble des états correspondant contient A, mais aussi $post(A)$, $post(post(A))$, etc. La réunion de tous les successeurs, directs ou non, des états de l'ensemble A,[6] forme un ensemble noté $sin[A;\mathcal{S}]$; les éléments de cet ensemble sont tous les états accessibles à partir d'un élément de A, en un nombre quelconque d'étapes. Le symbole *sin* signifie

[4] On a toujours $\widetilde{pre}(E) \subset pre(E)$; on a l'égalité dans le cas des programmes déterministes (cf. [Sifakis 82] pour plus de détail).

[5] une assertion vérifiant les trois contraintes introduites dans le paragraphe revinvint.

[6] Nous assimilons dorénavant une assertion à l'ensemble des états qu'elle représente.

"*s*trongest *in*variant". Cette appellation est justifiée; d'une part, cet ensemble est bien un invariant car le successeur d'un état accessible est lui-même un état accessible. En outre, si I est un invariant adéquat, alors $sin[A;\mathcal{S}]$ est plus fort que I. En effet, l'ensemble I, étant un invariant, s'identifie à l'ensemble des états accessibles à partir de I; comme A est plus petit (plus fort) que I, l'ensemble des éléments accessibles à partir de A est aussi plus petit (plus fort) que l'ensemble des éléments accessibles à partir de I. En conclusion, s'il existe des invariants adéquats, alors $sin[A;\mathcal{S}]$ est forcément l'un d'entre eux, en fait, le plus fort d'entre eux. Sur le plan théorique, le calcul de $sin[A;\mathcal{S}]$ est donc un moyen simple et élégant de mettre en œuvre la méthode des invariants. Signalons aussi la méthode duale: si I est un invariant adéquat, I doit être inclus non seulement dans J mais aussi dans $\widetilde{pre}(J)$, $\widetilde{pre}(\widetilde{pre}(J))$, etc. Soit $win[\mathcal{S};J]$ l'ensemble des états à partir desquels seuls des états vérifiant J sont accessibles. Cet ensemble est bien un invariant, en fait le plus faible[7] qui permette de prouver que J est une propriété de sûreté du système $\mathcal{S}$. La condition initiale A est adéquate si et seulement si $A \subset win[\mathcal{S};J]$, c'est-à-dire si la formule $A \Rightarrow win[\mathcal{S};J]$ est valide.

La théorie du point fixe introduite dans le volume II a permis de caractériser le plus fort invariant et le plus faible invariant dans le cas particulier d'une simple boucle (§ II.5.4.2). Ces caractérisations se généralisent aux cas des programmes séquentiels et parallèles quelconques, déterministes ou non. Supposons un système formel $\mathcal{S}$, séquentiel ou parallèle, dont l'ensemble des transitions est $\mathcal{T}$; soit aussi une assertion B sur l'espace d'état $\Sigma_{\mathcal{S}}$. On pose

$$slp[B;\mathcal{S}] =_{def} \bigvee_{\tau\in\mathcal{T}} slp[B;\tau]\,,$$

$$wlp[\mathcal{S};B] =_{def} \bigwedge_{\tau\in\mathcal{T}} wlp[\tau;B]\,.$$

Les expressions $slp[B;\tau]$ et $wlp[\tau;B]$ ont été définies au paragraphe II.3.4.4. Rappelons seulement ici les propriétés importantes de ces prédicats; $slp[B;\tau]$ est la plus forte postcondition Q telle que le triplet de Hoare $\{B\}\tau\{Q\}$ soit vrai, tandis que $wlp[\tau;B]$ est la plus faible précondition P telle que le triplet de Hoare $\{P\}\tau\{B\}$ soit vrai. Une conséquence immédiate de ces propriétés est que, pour tout système $\mathcal{S}$, l'assertion $slp[B;\mathcal{S}]$ correspond à l'ensemble $post(B)$ des états successeurs d'un état vérifiant B, tandis que l'assertion $wlp[\mathcal{S};B]$ correspond à l'ensemble $\widetilde{pre}(B)$ des états dont tous les successeurs vérifient B.

Considérons alors des assertions A et J, représentant par exemple une condition initiale et une propriété de sûreté pour le système $\mathcal{S}$. On définit les fonctions

[7] "*win*" signifie "*w*eakest *in*variant".

$$F[A,\mathcal{S}](X) =_{def} (A \vee slp[X;\mathcal{S}]),$$
$$G[\mathcal{S},J](X) =_{def} (J \wedge wlp[\mathcal{S};X]).$$

Ces fonctions sont des transformations de l'ensemble $\mathcal{P}(\Sigma)$ des assertions. Sur cet ensemble, on vérifie facilement que la fonction $F[A,\mathcal{S}]$ est $\vee$-additive et que la fonction $G[\mathcal{S},J]$ est $\wedge$-additive, ce qui signifie que pour toute famille d'assertions $\{X_k : k \in K\}$ on a

$$F[A,\mathcal{S}](\bigvee_{k\in K} X_k) \equiv \bigvee_{k\in K} F[A,\mathcal{S}](X_k),$$
$$G[\mathcal{S},J](\bigwedge_{k\in K} X_k) \equiv \bigwedge_{k\in K} G[\mathcal{S},J](X_k).$$

Une conséquence immédiate de ces résultats est que ces fonctions sont continues, l'une pour l'ordre induit par l'implication, l'autre pour l'ordre inverse (il suffit de considérer le cas particulier où la famille d'assertions est une chaîne; cf. § II.5.2.5), ce qui permet de calculer le plus petit (le plus fort) point fixe de la fonction $F[A,\mathcal{S}]$ et le plus grand (le plus faible) point fixe de la fonction $G[\mathcal{S},J]$ par les formules

$$sin[A,\mathcal{S}] = \mathrm{pppf}(F[A,\mathcal{S}]) = \bigvee_{n\in\mathbb{N}} F[A,\mathcal{S}]^n(false),$$
$$win[\mathcal{S},J] = \mathrm{pgpf}(G[\mathcal{S},J]) = \bigwedge_{n\in\mathbb{N}} G[\mathcal{S},J]^n(true).$$

Le fait que le plus petit point fixe de $F[A,\mathcal{S}]$ et le plus grand point fixe de $G[\mathcal{S},J]$ correspondent respectivement au plus fort invariant et au plus faible invariant est une conséquence immédiate des résultats suivants: l'assertion $F[A,\mathcal{S}]^n(false)$ est vraie dans un état donné σ si et seulement s'il existe un état σ_0 vérifiant A tel que σ soit accessible à partir de σ_0 en moins de n étapes. De même, l'assertion $G[\mathcal{S},J]^n(true)$ est vraie dans un état donné σ si et seulement s'il n'est pas possible d'accéder en moins de n étapes, au départ de σ, à un état ne vérifiant pas J.

13.2.2. *Application : programme séquentiel*

A titre d'exemple, nous essayons de calculer $sin[A;\mathcal{S}]$ où $\mathcal{S}$ est le programme de calcul du plus grand commun diviseur et où A est l'assertion

$$at\ \ell_0 \wedge x = x_0 \wedge y = y_0.$$

On suppose que x_0 et y_0 sont des entiers strictement positifs, non autrement précisés. L'idée est d'obtenir le plus fort invariant $sin[A;\mathcal{S}]$ comme la limite de la disjonction des assertions $F[A,\mathcal{S}]^n(false)$. On remarque d'emblée que toute exécution de la transition α est d'office suivie par l'exécution de la transition δ; on peut donc fusionner les deux transitions. De même, on peut fusionner β

et ε, et γ et ζ. Opérer ces fusions accélèrera la convergence et simplifiera les calculs symboliques. On considère donc le système suivant :[8]

$$P = \{\ell_0, \ell_e\};$$

$$\mathcal{M} = \{x, y, g : \text{entiers strictement positifs}\};$$

$$\begin{aligned}\mathcal{T} = \{\, & a : \ (\ell_0,\ x > y \longrightarrow x := x - y,\ \ell_0)\,,\\ & b : \ (\ell_0,\ x < y \longrightarrow y := y - x,\ \ell_0)\,,\\ & c : \ (\ell_0,\ x = y \longrightarrow g := x,\ \ell_e)\,\}\,.\end{aligned}$$

On calcule alors les premiers termes de la suite $(F[A, \mathcal{S}]^n(\mathit{false}) : n \in \mathbb{N})$:

$$\begin{aligned}
F^0 &: \ \mathit{false}\,,\\
F^1 &: \ (\mathit{at}\ \ell_0 \wedge x = x_0 \wedge y = y_0)\,,\\
F^2 &: \ (\mathit{at}\ \ell_0 \wedge x = x_0 \wedge y = y_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_0 \wedge x_0 > y_0 \wedge x = x_0 - y_0 \wedge y = y_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_0 \wedge y_0 > x_0 \wedge x = x_0 \wedge y = y_0 - x_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_e \wedge x_0 = y_0 = x = y = g)\,,\\
F^3 &: \ (\mathit{at}\ \ell_0 \wedge x = x_0 \wedge y = y_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_0 \wedge x_0 > y_0 \wedge x = x_0 - y_0 \wedge y = y_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_0 \wedge x_0 > 2y_0 \wedge x = x_0 - 2y_0 \wedge y = y_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_0 \wedge x_0 > y_0 > x_0 - y_0 \wedge x = x_0 - y_0 \wedge y = 2y_0 - x_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_0 \wedge y_0 > x_0 \wedge x = x_0 \wedge y = y_0 - x_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_0 \wedge y_0 > 2x_0 \wedge x = x_0 \wedge y = y_0 - 2x_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_0 \wedge y_0 > x_0 > y_0 - x_0 \wedge x = 2x_0 - y_0 \wedge y = y_0 - x_0) \ \vee\\
&\quad\ (\mathit{at}\ \ell_e \wedge x_0 = y_0 = x = y = g) \ \vee\\
&\quad\ (\mathit{at}\ \ell_e \wedge x_0 - y_0 = y_0 = x = y = g) \ \vee\\
&\quad\ (\mathit{at}\ \ell_e \wedge x_0 = y_0 - x_0 = x = y = g)\,,\\
F^4 &: \ \ldots
\end{aligned}$$

On devine aisément que cette méthode de construction de l'invariant, bien que théoriquement élégante, est peu intéressante en pratique; la taille des formules successives augmente trop rapidement. Il n'est pas surprenant qu'une méthode générale de construction d'invariant soit en pratique difficilement applicable. En effet, l'invariant décrit très précisément le programme et permet d'établir la majorité de ses propriétés intéressantes. Une technique automatique de construction d'invariant équivaudrait presque à une méthode automatique de

[8] qui, pour l'utilisateur extérieur, ne se distingue en rien du système initial.

preuve de programme, qui inclurait nécessairement une méthode automatique de preuve des théorèmes de l'arithmétique, laquelle ne peut exister, d'après un résultat célèbre dû à Gödel.

D'un point de vue plus concret, un programme tel que celui d'Euclide est basé sur une idée précise; sans la connaissance de celle-ci, on peut tester le programme et l'utiliser, mais pas réellement le comprendre ni démontrer ses propriétés. L'idée d'Euclide peut s'énoncer comme suit :

Soient deux entiers naturels a et b tels que $a > b > 0$.
On a toujours $gcd(a, b) = gcd(a - b, b)$.

Cette propriété est facile à démontrer mais la difficulté est ailleurs : pour obtenir l'algorithme d'Euclide, il faut *découvrir* cette propriété et prendre conscience de son caractère opérationnel, c'est-à-dire de la possibilité d'en tirer un algorithme de calcul pour le plus grand commun diviseur de deux entiers naturels.

Remarque. L'algorithme classique d'Euclide est plus efficace et aussi un peu plus complexe que celui présenté ici. Il exploite le fait que, si a est plus grand que b, on a non seulement $\gcd(a - b, b) = \gcd(a, b)$, mais aussi $\gcd(a - kb, b) = \gcd(a, b)$, pour tout entier k et en particulier pour le plus grand entier k tel que $a - kb$ soit positif.

13.2.3. *Application : programme parallèle*

Découvrir les invariants appropriés de programmes classiques requiert de la créativité; simuler la créativité est sans doute l'aspect le plus difficile de l'intelligence artificielle et nous ne l'aborderons pas ici. Par contre, on peut étudier la possibilité de construire systématiquement des invariants dans les cas où on peut raisonnablement penser que cette tâche ne requiert pas de réelle créativité. C'est fréquemment le cas dans le domaine des algorithmes parallèles non destinés à mettre en œuvre un résultat mathématique non trivial, mais plutôt à permettre à différents processus de coopérer harmonieusement. De ce point de vue, les algorithmes d'exclusion mutuelle étudiés dans le chapitre 4 du volume II constituent un exemple typique. Nous avions montré comment la méthode des invariants se prêtait très bien à la vérification des propriétés de tels programmes et nous avions aussi argumenté sur la nécessité réelle de cette démarche de démonstration formelle.[9]

A titre de premier exemple nous considérons la version simple de l'algorithme de Peterson représentée à la figure 13.1. Dans le volume précédent (§ II.4.4.6) on avait utilisé la technique des invariants pour montrer que, moyennant la condition initiale

[9] Rappelons seulement ici que ces algorithmes jouent un rôle critique dans les systèmes informatiques modernes et que les procédures de test, relativement acceptables dans le cas des programmes séquentiels classiques, deviennent inopérantes dans le cas des systèmes parallèles.

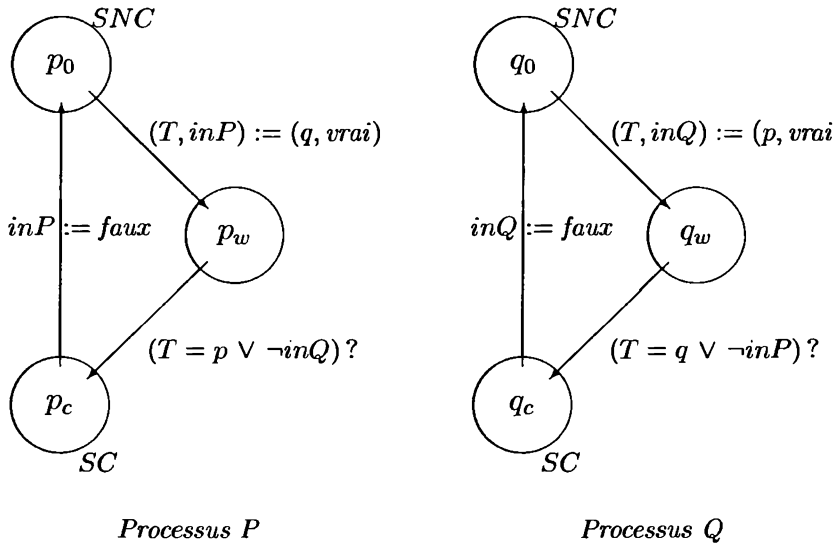


Figure 13.1 *Version simplifiée de l'algorithme de Peterson*

$$A \ : \ at\, p_0 q_0 \wedge \neg inP \wedge \neg inQ \wedge (T = p \vee T = q)\,,$$

l'algorithme était correct par rapport à la propriété d'exclusion mutuelle, formalisée par l'assertion

$$J \ : \ \neg at\, p_c q_c\,.$$

Nous montrons comment l'invariant peut être obtenu au moyen de la fonctionnelle $F[A, \mathcal{S}]$, selon la technique introduite au paragraphe précédent. Les premiers termes de la suite d'approximations sont :

$$F^0 \ : \ false\,,$$

$$F^1 \ : \ (at\, p_0 q_0 \wedge \neg inP \wedge \neg inQ \wedge (T = p \vee T = q))\,,$$

$$\begin{aligned} F^2 \ : \ & F^1 \ \vee \\ & (at\, p_w q_0 \wedge inP \wedge \neg inQ \wedge T = q) \ \vee \\ & (at\, p_0 q_w \wedge \neg inP \wedge inQ \wedge T = p)\,, \end{aligned}$$

$$\begin{aligned} F^3 \ : \ & F^2 \ \vee \\ & (at\, p_w q_w \wedge inP \wedge inQ \wedge (T = p \vee T = q)) \ \vee \\ & (at\, p_c q_0 \wedge inP \wedge \neg inQ \wedge T = q) \ \vee \\ & (at\, p_0 q_c \wedge \neg inP \wedge inQ \wedge T = p)\,, \end{aligned}$$

$$
\begin{aligned}
F^4 \;:\;\; & F^3 \;\vee \\
& (at\ p_c q_w \wedge inP \wedge inQ \wedge T = p) \;\vee \\
& (at\ p_w q_c \wedge inP \wedge inQ \wedge T = q)\,,
\end{aligned}
$$

$$
F^5 \;:\;\; F^4\,.
$$

On peut récrire cet invariant de manière plus lisible, en précisant les valeurs des variables en fonction de celles des points de contrôle. L'inconvénient de cette technique est qu'elle allonge la formule. On obtient ainsi :

$$
\begin{aligned}
& (at\ p_0 q_0 \wedge \neg inP \wedge \neg inQ \wedge (T = p \vee T = q)) \\
\vee\ & (at\ p_w q_0 \wedge inP \wedge \neg inQ \wedge T = q) \\
\vee\ & (at\ p_0 q_w \wedge \neg inP \wedge inQ \wedge T = p) \\
\vee\ & (at\ p_w q_w \wedge inP \wedge inQ \wedge (T = p \vee T = q)) \\
\vee\ & (at\ p_c q_0 \wedge inP \wedge \neg inQ \wedge T = q) \\
\vee\ & (at\ p_0 q_c \wedge \neg inP \wedge inQ \wedge T = p) \\
\vee\ & (at\ p_c q_w \wedge inP \wedge inQ \wedge T = p) \\
\vee\ & (at\ p_w q_c \wedge inP \wedge inQ \wedge T = q)\,.
\end{aligned}
$$

On peut aussi lui donner une forme plus concise :

$$
\begin{aligned}
I \;:\;\; & (at\ p_0 \equiv \neg inP) \wedge (at\ q_0 \equiv \neg inQ) \\
\wedge\ & (at\ p_w \Rightarrow (at\ q_w \vee T = q)) \wedge (at\ q_w \Rightarrow (at\ p_w \vee T = p)) \\
\wedge\ & (at\ p_c \Rightarrow (at\ q_w \ominus T = q)) \wedge (at\ q_c \Rightarrow (at\ p_w \ominus T = p))\,.
\end{aligned}
$$

(Rappelons que le symbole $\ominus$ désigne le "ou" exclusif.)

L'invariant obtenu est le plus fort, mais pas le seul convenable. En particulier, l'invariant utilisé pour vérifier l'algorithme dans le volume II était

$$
\begin{aligned}
I' \;:\;\; & (at\ p_0 \equiv \neg inP) \wedge (at\ q_0 \equiv \neg inQ) \\
\wedge\ & (at\ p_c \Rightarrow (T = p \vee \neg inQ)) \wedge (at\ q_c \Rightarrow (T = q \vee \neg inP))\,,
\end{aligned}
$$

dont on peut vérifier qu'il est strictement plus faible que le précédent. Les deux invariants sont adéquats puisque $I[at\ p_c q_c]$ et $I'[at\ p_c q_c]$ se réduisent tous deux à *false*.

Par rapport à l'algorithme d'Euclide, on peut observer deux phénomènes intéressants : il n'y a pas croissance démesurée de la taille des formules, et la suite des approximations est stationnaire : la limite est atteinte à la quatrième itération. La stationnarité était prévisible; dans la mesure où l'espace d'état

$$
\Sigma \;=\; \{p_0, p_w, p_c\} \times \{q_0, q_w, q_c\} \times \{vrai, faux\} \times \{vrai, faux\} \times \{p, q\}
$$

est fini (il comporte $3^2 2^3 = 72$ états), l'ensemble des assertions est également fini (2^{72} éléments) et toute suite monotone est nécessairement stationnaire.

Si on cherche maintenant à valider la version complète de l'algorithme de Peterson, obtenue en remplaçant les deux transitions

$$(p_0\,,\ (T, inP) := (q, vrai)\,,\ p_w)\,,\quad (q_0\,,\ (T, inQ) := (p, vrai)\,,\ q_w)$$

par les quatre transitions

$$(p_0\,,\ inP := vrai\,,\ p_i)\,,\quad (q_0\,,\ inQ := vrai\,,\ q_i)\,,$$
$$(p_i\,,\ T := q\,,\ p_w)\,,\qquad (q_i\,,\ T := p\,,\ q_w)\,,$$

il faut utiliser un autre invariant. On ne doit cependant pas recommencer tout le travail de construction; il suffit de remarquer que le nouvel invariant sera nécessairement plus faible que l'ancien, ce qui revient à relancer le processus de construction en prenant comme nouvelle condition initiale l'assertion

$$I \wedge \neg at\, p_i \wedge \neg at\, q_i\,.$$

On obtient aisément les approximations suivantes:

$$F^0 : \ false\,,$$

$$\begin{aligned} F^1 : \ & (at\, p_0 q_0 \wedge \neg inP \wedge \neg inQ \wedge (T = p \vee T = q)) \\ \vee \ & (at\, p_w q_0 \wedge inP \wedge \neg inQ \wedge T = q) \\ \vee \ & (at\, p_0 q_w \wedge \neg inP \wedge inQ \wedge T = p) \\ \vee \ & (at\, p_w q_w \wedge inP \wedge inQ \wedge (T = p \vee T = q)) \\ \vee \ & (at\, p_c q_0 \wedge inP \wedge \neg inQ \wedge T = q) \\ \vee \ & (at\, p_0 q_c \wedge \neg inP \wedge inQ \wedge T = p) \\ \vee \ & (at\, p_c q_w \wedge inP \wedge inQ \wedge T = p) \\ \vee \ & (at\, p_w q_c \wedge inP \wedge inQ \wedge T = q)\,, \end{aligned}$$

$$\begin{aligned} F^2 : \ & F^1 \\ \vee \ & (at\, p_0 q_i \wedge \neg inP \wedge inQ \wedge (T = p \vee T = q)) \\ \vee \ & (at\, p_i q_0 \wedge inP \wedge \neg inQ \wedge (T = p \vee T = q)) \\ \vee \ & (at\, p_w q_i \wedge inP \wedge inQ \wedge T = q) \\ \vee \ & (at\, p_i q_w \wedge inP \wedge inQ \wedge T = p) \\ \vee \ & (at\, p_c q_i \wedge inP \wedge inQ \wedge T = q) \\ \vee \ & (at\, p_i q_c \wedge inP \wedge inQ \wedge T = p)\,, \end{aligned}$$

$$F^3 : \ F^2 \vee (at\, p_i q_i \wedge inP \wedge inQ \wedge (T = p \vee T = q))$$

$$F^4 : \ F^3\,.$$

Ici aussi, l'invariant obtenu est le plus fort mais d'autres invariants adéquats peuvent exister.

13.3. Construction de programme

L'exemple de l'algorithme de Peterson montre que les invariants peuvent se construire de manière relativement systématique. Il serait pourtant intéressant

de faire intervenir cette technique lors de la construction du programme lui-même, ce qui reviendrait à vérifier chaque étape du développement et à détecter les erreurs éventuelles dès leur apparition. Nous montrons sur un exemple simple que cette stratégie est possible en pratique, mais que certaines adaptations de l'invariant requièrent quand même un peu de créativité.

13.3.1. *Un protocole de transfert de données*

Un protocole de transfert de données doit assurer une transmission d'information fiable au travers d'un canal de transmission qui ne l'est pas, c'est-à-dire qui peut retarder, corrompre ou éliminer les messages qu'il véhicule. On suppose que la station recevant les messages est capable de détecter et d'écarter les messages corrompus.

L'information à transmettre est représentée par une séquence $X =_{def} (X[n] : n = 1, 2, \ldots)$ de messages; une séquence analogue Y représente la partie correctement transmise de X. L'émetteur sera inévitablement amené à transmettre certains messages plusieurs fois; on suppose que chaque message $X[n]$ comporte le numéro n permettant de l'identifier.

13.3.1.1. *La version initiale*

Dans un premier temps on ignore complètement les problèmes de transmission; l'émission et la réception correcte et instantanée d'un message est modélisée par une affectation. Notre système initial $\mathcal{S}_0$ est

$$\mathcal{P}_0 = \emptyset;$$
$$\mathcal{M}_0 = \{HS : nat;\ X, Y : array[nat]\ of\ string\};$$
$$\mathcal{T}_0 = \{((HS, Y[HS+1]) := (HS+1, X[HS+1]))\}.$$

L'index HS (pour "Highest Sent") identifie par son numéro le dernier message transmis. L'invariant de $\mathcal{S}_0$ est simplement

$$I_0 =_{def} \quad \forall s\,(1 \leq s \leq HS \Rightarrow Y[s] = X[s]) \ \wedge$$
$$\forall s\,(HS < s \Rightarrow Y[s] = NIL).$$

13.3.1.2. *Perte de messages*

La possibilité de perte, de retard ou de corruption d'un message implique que le dernier message reçu n'est pas nécessairement le dernier émis. Une nouvelle variable LR (pour "Last Received") est introduite pour préparer la modélisation de cette possibilité. Cela conduit au système $\mathcal{S}_1$, dont la seule transition est l'affectation

$$(HS, LR, Y[HS+1]) := (HS+1, LR+1, X[HS+1]).$$

Le nouvel invariant I_1 est $(I_0 \wedge HS = LR)$. On peut maintenant modéliser la possibilité de perte ou corruption par le système $\mathcal{S}_2$ où $\mathcal{P}_2 = \mathcal{P}_1$, $\mathcal{M}_2 = \mathcal{M}_1$; l'ensemble $\mathcal{T}_2$ des transitions comporte les quatre affectations conditionnelles suivantes :

$$(LR{=}HS \longrightarrow (HS, LR, Y[HS{+}1]) := (HS{+}1, LR{+}1, X[HS{+}1]))\,,$$
$$(LR{=}HS \longrightarrow HS := HS + 1)\,,$$
$$(LR{<}HS \longrightarrow (LR, Y[HS]) := (LR + 1, X[HS]))\,,$$
$$(LR{<}HS \longrightarrow \mathit{skip})\,.$$

La première transition modélise la transmission correcte d'un nouveau message et la deuxième, sa transmission incorrecte; la troisième transition modélise la retransmission fructueuse d'un message déjà (incorrectement) transmis, tandis que la quatrième représente une retransmission infructueuse. Comme on suppose qu'un message corrompu est reconnu comme tel, on ne distingue pas entre message corrompu et message perdu.

On calcule facilement le nouvel invariant I_2 :

$$F^0 : \mathit{false}\,,$$

$$F^1 : \forall s\,(1 \leq s \leq HS \Rightarrow Y[s] = X[s]) \wedge \forall s\,(HS < s \Rightarrow Y[s] = NIL) \wedge HS = LR\,,$$

$$F^2 : \forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge \forall s\,(LR < s \Rightarrow Y[s] = NIL) \wedge LR \leq HS \leq LR + 1\,,$$

$$F^3 : F^2$$

13.3.1.3. *Plus de parallélisme*

Il est clair que, dans une version réaliste du protocole, l'index HS sera géré par la station émettrice tandis que l'index LR sera géré par la station réceptrice, de manière relativement indépendante. Pour préparer une telle version il convient de scinder les transitions qui altèrent les deux index à la fois. La station réceptrice détecte la possibilité d'augmenter son index quand le message suivant celui-ci à été correctement reçu. Ceci conduit au système $\mathcal{S}_3$, semblable à $\mathcal{S}_2$ sauf que l'ensemble $\mathcal{T}_3$ des transitions est :

$$1 : (LR = HS \longrightarrow (HS, Y[HS + 1]) := (HS + 1, X[HS + 1]))\,,$$
$$2 : (LR = HS \longrightarrow HS := HS + 1)\,,$$
$$3 : (LR < HS \longrightarrow Y[HS] := X[HS])\,,$$
$$4 : (LR < HS \longrightarrow \mathit{skip})\,,$$
$$5 : (Y[LR + 1] \neq NIL \longrightarrow LR := LR + 1)\,\}\,.$$

On calcule facilement le nouvel invariant I_3 :

$$F^0 : \; \mathit{false},$$

$$F^1 : \; \forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge \forall s\,(LR < s \Rightarrow Y[s] = NIL) \wedge LR \leq HS \leq LR + 1,$$

$$F^2 : \; \forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge (Y[HS] = X[HS] \vee Y[HS] = NIL) \wedge \forall s\,(HS < s \Rightarrow Y[s] = NIL) \wedge LR \leq HS \leq LR + 1,$$

$$F^3 : \; F^2.$$

La présente version ne comporte plus de transition *altérant* simultanément les deux index mais certaines transitions *testent* encore simultanément ces index. Pour éviter cela, on peut doter la station émettrice d'un nouvel index LA (pour "Last Acknowledged"), copie de la variable LR; dans un premier temps on suppose que la variable et sa copie évoluent de façon synchrone. Cela donne lieu à la version $\mathcal{S}_4$, dont les transitions sont :

$$1 : (LA = HS \longrightarrow (HS, Y[HS+1]) := (HS+1, X[HS+1])),$$
$$2 : (LA = HS \longrightarrow HS := HS + 1),$$
$$3 : (LA < HS \longrightarrow Y[HS] := X[HS]),$$
$$4 : (LA < HS \longrightarrow \mathit{skip}),$$
$$5 : (Y[LR+1] \neq NIL \longrightarrow (LA, LR) := (LR+1, LR+1))\}.$$

L'invariant est naturellement $I_4 =_{def} (I_3 \wedge LA = LR)$. En pratique toutefois, la copie LA, gérée par l'émetteur, ne pourra rester en parfait synchronisme avec l'index LR, géré par le récepteur. Ce dernier devra, à chaque arrivée d'un message, envoyer un accusé de réception à l'émetteur, ce qui permettra à ce dernier d'actualiser LA. Cet accusé de réception, comme tout message, est susceptible de retard, d'altération ou de perte. On obtient ainsi la nouvelle version $\mathcal{S}_5$, dont la mémoire est :

$$\mathcal{M}_5 = \{LA, LR, HS : \mathit{nat};\; X, Y : \mathit{array}[\mathit{nat}]\ \mathit{of}\ \mathit{string}\};$$

et dont l'ensemble $\mathcal{T}_5$ des transitions se compose de :

$$
\begin{aligned}
&1 : (LA = HS \longrightarrow (HS, Y[HS+1]) := (HS+1, X[HS+1])),\\
&2 : (LA = HS \longrightarrow HS := HS+1),\\
&3 : (LA < HS \longrightarrow Y[HS] := X[HS]),\\
&4 : (LA < HS \longrightarrow skip),\\
&5 : (Y[LR+1] \neq NIL \longrightarrow (LA, LR) := (LR+1, LR+1)),\\
&6 : (Y[LR+1] \neq NIL \longrightarrow LR := LR+1),\\
&7 : (Y[LR+1] = NIL \longrightarrow LA := LR),\\
&8 : (Y[LR+1] = NIL \longrightarrow skip))\}.
\end{aligned}
$$

La technique des approximations successives permet d'adapter l'invariant une fois de plus; on obtient :

$$
\begin{aligned}
I_5 \equiv\ & (LA \leq LR \leq HS \leq LA+1) \wedge\\
& \forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge\\
& (Y[HS] = X[HS] \vee Y[HS] = NIL) \wedge\\
& \forall s\,(HS < s \Rightarrow Y[s] = NIL)\,.
\end{aligned}
$$

Les transitions 1 et 2 représentent la transmission d'un nouveau message, quand le dernier message transmis a été correctement "acquitté".[10] Cette transmission peut réussir (transition 1) ou échouer (transition 2). Si le message $X[HS]$ n'a pas été acquitté endéans un certain délai, il est retransmis et cette retransmission peut réussir (transition 3) ou échouer (transition 4). Quand un nouveau message est reçu et qu'aucune altération n'est détectée, un accusé de réception est envoyé, qui peut être correctement reçu (transition 5) ou non (transition 6). L'accusé de réception est retransmis, avec succès (transition 7) ou non (transition 8), jusqu'à bonne réception du message suivant.

13.3.2. *Le principe de la fenêtre de Stenning*

13.3.2.1. *L'idée générale*

Dans notre version, un synchronisme relativement étroit est maintenu entre l'émetteur et le récepteur, puisque le premier n'a au plus qu'un message d'avance sur le second. Il peut être opportun de permettre un plus grand écart et de prévoir une "fenêtre" de messages en cours de transmission ou d'acquittement. Soit $W \geq 1$ la taille maximale (finie) de cette fenêtre. Les messages candidats à une retransmission sont $X[r] : LA < r \leq HS$. La version précédente $\mathcal{S}_5$ correspondait au cas $W = 1$; la généralisation proposée ici (cf. [Stenning 76]), requérant moins de synchronisme, donne souvent lieu à une meilleure performance globale.

[10] Acquitter un message signifie envoyer un accusé de réception; un message est acquitté quand l'accusé de réception a été reçu par l'émetteur.

13.3.2.2. *Le développement*

Une étape préliminaire consiste en l'introduction de l'index de fenêtre r; cela conduit à la version $\mathcal{S}_6$, telle que $\mathcal{M}_6 = \mathcal{M}_5 \cup \{r : nat\}$. L'invariant I_6 est identique à I_5. On peut maintenant introduire l'idée de Stenning dans la version $\mathcal{S}_7$, dont l'ensemble $\mathcal{T}_7$ des transitions est :

1. $(HS - LA < W \longrightarrow (HS, Y[HS+1]) := (HS+1, X[HS+1]))$,
2. $(HS - LA < W \longrightarrow HS := HS + 1)$,
3. $(LA < r \leq HS \longrightarrow (Y[r], r) := (X[r], r + 1))$,
4. $(LA < r \leq HS \longrightarrow r := r + 1)$,
5. $(\neg(LA < r \leq HS) \longrightarrow r := LA + 1)$,
6. $(Y[LR + 1] \neq NIL \longrightarrow (LA, LR) := (LR + 1, LR + 1))$,
7. $(Y[LR + 1] \neq NIL \longrightarrow LR := LR + 1)$,
8. $(Y[LR + 1] = NIL \longrightarrow LA := LR)$,
9. $(Y[LR + 1] = NIL \longrightarrow skip)\}$.

Les transitions 1 et 2 correspondent à une première transmission d'un message (réussie ou non); les transitions 3, 4 et 5 modélisent la retransmission des messages de la fenêtre; les transitions 6 et 7 correspondent à la réception suivie de l'envoi (réussi ou non) d'un accusé de réception; enfin, les transitions 8 et 9 mettent en œuvre la retransmission des accusés de réception.

L'adaptation de l'invariant par la méthode habituelle n'est pas tout à fait évidente. On a les approximations suivantes :

$$F^0 : \quad false,$$

$$\begin{aligned} F^1 : \quad & (LA \leq LR \leq HS \leq LA + 1) \wedge \\ & \forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge \\ & (Y[HS] = X[HS] \vee Y[HS] = NIL) \wedge \\ & \forall s\,(HS < s \Rightarrow Y[s] = NIL]), \end{aligned}$$

$$\begin{aligned} F^2 : \quad & (LA \leq LR \leq HS \leq LA + 2) \wedge \\ & \forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge \\ & \forall s\,(LR < s \leq HS \Rightarrow (Y[s] = X[s] \vee Y[s] = NIL])) \wedge \\ & \forall s\,(HS < s \Rightarrow Y[s] = NIL]), \end{aligned}$$

$$\begin{aligned} F^3 : \quad & (LA \leq LR \leq HS \leq LA + 3) \wedge \\ & \forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge \\ & \forall s\,(LR < s \leq HS \Rightarrow (Y[s] = X[s] \vee Y[s] = NIL])) \wedge \\ & \forall s\,(HS < s \Rightarrow Y[s] = NIL]), \end{aligned}$$

On devine aisément que, pour tout $n < W$, on aura

$$\begin{aligned} F^n : \quad & (LA \leq LR \leq HS \leq LA + n) \wedge \\ & \forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge \\ & \forall s\,(LR < s \leq HS \Rightarrow (Y[s] = X[s] \vee Y[s] = NIL])) \wedge \\ & \forall s\,(HS < s \Rightarrow Y[s] = NIL])\,, \end{aligned}$$

et, pour tout $n \geq W$, on aura

$$\begin{aligned} F^n : \quad & (LA \leq LR \leq HS \leq LA + W) \wedge \\ & \forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge \\ & \forall s\,(LR < s \leq HS \Rightarrow (Y[s] = X[s] \vee Y[s] = NIL])) \wedge \\ & \forall s\,(HS < s \Rightarrow Y[s] = NIL])\,, \end{aligned}$$

ce qui donne le nouvel invariant I_7. Toutefois, si on essaie de mécaniser ce développement, on rencontre deux obstacles. Le premier apparaît dès que l'on calcule F^2 à partir de F^1. L'opérateur humain effectue sans y penser des simplifications telles que la réduction de $HS \leq (LA+1)+1$ en $HS \leq LA+2$; le système automatique ne fera cette simplification que si elle a été programmée. La difficulté n'est pas de programmer cette simplification mais de faire une liste exhaustive de toutes les simplifications à programmer. Le second obstacle est d'automatiser le raisonnement inductif permettant à l'opérateur humain de deviner que si F^n comporte $HS \leq LA + n$ pour $n = 1, 2, 3$, alors il est envisageable que F^n comporte $HS \leq LA+n$ pour tout n. Ici aussi, la difficulté ne consiste pas à vérifier ou à infirmer ce fait, mais simplement à le considérer.

13.3.3. *Représentation des processus*

Jusqu'ici nous n'avons pas explicité quels processus intervenaient dans quelles transitions, même si, informellement, l'adaptation de certaines variables est attribuée à la station émettrice et l'adaptation d'autres variables à la station réceptrice.

13.3.3.1. *Identification des processus*

On conçoit aisément qu'introduire deux processus, un émetteur et un récepteur, constitue une solution acceptable. Toutefois, scinder les deux stations en plusieurs processus peut améliorer le parallélisme des tâches et conduire à de meilleures performances globales.

La station émettrice a en fait trois fonctions: elle envoie de nouveaux messages, elle retransmet des messages pour lesquels elle n'a pas reçu d'accusé de réception au bout d'un certain délai,[11] et elle reçoit les accusés de

[11] Ce délai n'est pas modélisé ni discuté ici.

réception. Ces trois tâches seront atribuées à trois processus distincts, nommés respectivement A, B et C. De même, la station réceptrice a deux rôles, recevoir les messages (processus D) et envoyer les accusés de réception (processus E). Dans la version suivante, chaque processus comporte une seule étiquette.[12] L'ensemble des processus du système $\mathcal{S}_8$ est donc

$$\mathcal{P}_8 = \{A, B, C, D, E\}\text{, où}$$
$$A = \{A_0\},\ B = \{B_0\},\ C = \{C_0\},\ D = \{D_0\},\ E = \{E_0\}.$$

L'ensemble $\mathcal{T}_8$ des transitions sera

$$1 : (A_0D_0,\ HS - LA < W \longrightarrow (HS, Y[HS+1]) := (HS+1, X[HS+1]),\ A_0D_0),$$
$$2 : (A_0,\ HS - LA < W \longrightarrow HS := HS+1,\ A_0),$$
$$3 : (B_0D_0,\ LA < r \leq HS \longrightarrow (Y[r], r) := (X[r], r+1),\ B_0D_0),$$
$$4 : (B_0,\ LA < r \leq HS \longrightarrow r := r+1,\ B_0),$$
$$5 : (B_0,\ \neg(LA < r \leq HS) \longrightarrow r := LA+1,\ B_0),$$
$$6 : (C_0E_0,\ Y[LR+1] \neq NIL \longrightarrow (LA, LR) := (LR+1, LR+1),\ C_0E_0),$$
$$7 : (E_0,\ Y[LR+1] \neq NIL \longrightarrow LR := LR+1,\ E_0),$$
$$8 : (C_0E_0,\ Y[LR+1] = NIL \longrightarrow LA := LR,\ C_0E_0),$$
$$9 : (E_0,\ Y[LR+1] = NIL \longrightarrow skip,\ E_0)\}.$$

On notera que les transitions modélisant les transmissions réussies impliquent deux processus (l'envoyeur et le destinataire) tandis que celles modélisant des échecs impliquent un seul processus (l'envoyeur).[13]

L'invariant I_8 de cette version distribuée est simplement I_7 car l'introduction explicite des processus n'a pas d'influence sur l'ensemble des exécutions du système.

13.3.3.2. *Communication asynchrone*

Dans la version précédente, on utilise le mode de communication *synchrone*; une même transition modélise l'émission et la réception (fructueuse ou non) d'un message ou d'un acquit. En pratique, on préfère souvent le mode de communication *asynchrone*, dans lequel un certain délai peut séparer l'émission de la réception; pendant ce délai, l'envoyeur peut poursuivre sa tâche, ce qui améliore les performances globales du système. En toute généralité, la transmission synchrone d'une valeur x provenant d'un processus p vers une variable y appartenant à un processus q se modélise par la transition

[12] Rappelons qu'un système peut ne pas comporter de processus, mais que chaque processus comporte au moins un point de contrôle.

[13] L'envoyeur est un processus de la station émettrice pour les messages, et un processus de la station réceptrice pour les accusés de réception.

$$(p_1 q_1,\ y := x,\ p_2 q_2),$$

si le canal de transition est fiable, et par les deux transitions

$$(p_1 q_1,\ y := x,\ p_2 q_2),$$
$$(p_1,\ skip,\ p_2),$$

si le canal n'est pas fiable (on suppose cependant que le processus q est capable de rejeter un message corrompu). En mode asynchrone, la communication via un canal C supposé fiable se modélise par les deux transitions suivantes :

$$(p_1,\ C := x,\ p_2),$$
$$(q_1,\ C \neq NIL \longrightarrow (y, C) := (C, NIL),\ q_2).$$

Si le canal C n'est pas fiable, on utilise les trois transitions suivantes :

$$(p_1,\ C := x,\ p_2),$$
$$(q_1,\ C \neq NIL \longrightarrow (y, C) := (C, NIL),\ q_2),$$
$$(\emptyset,\ C := NIL,\ \emptyset).$$

La dernière transition n'implique aucun processus, parce qu'elle n'est le fait ni de l'émetteur ni du récepteur, mais plutôt du milieu de transmission.

Remarque. D'autres techniques de communication sont utilisables mais les introduire sortirait de notre propos.

On introduit donc une variable MB (pour "Mail Buffer") pour représenter le canal de transmission des messages. La composante $MB.1$ contient le numéro d'identification du message, la composante $MB.2$ contient le message proprement dit. Cela donne lieu à la version $\mathcal{S}_9$ dont l'invariant I_9 précisera le comportement de la nouvelle variable : elle contient un message, précédé de son numéro, ou elle est vide, ce que l'on représente par la valeur NIL.[14] On a donc

$$I_9 =_{def} I_8 \wedge (MB = NIL \vee \exists s\,[(1 \leq s \leq HS) \wedge MB = (s, X[s])]).$$

On peut à présent introduire la communication asynchrone proprement dite. Cela conduit au système $\mathcal{S}_{10}$, dont l'ensemble de transitions $\mathcal{T}_{10}$ comporte :

$$1 : (A_0,\ HS - LA < W \longrightarrow (HS, MB) := (HS + 1, (HS + 1, X[HS + 1])),\ A_0),$$
$$2 : (B_0,\ LA < r \leq HS \longrightarrow (MB, r) := ((r, X[r]), r + 1),\ B_0),$$
$$3 : (B_0,\ \neg(LA < r \leq HS) \longrightarrow r := LA + 1,\ B_0),$$
$$4 : (D_0,\ MB \neq NIL \longrightarrow (Y[MB.1], MB) := (MB.2, NIL),\ D_0),$$

[14] Cette valeur représente aussi un message corrompu.

$5 : (C_0E_0,\ Y[LR+1] \neq NIL \longrightarrow$
$(LA, LR) := (LR+1, LR+1)\ ,\ C_0E_0)\ ,$
$6 : (E_0,\ Y[LR+1] \neq NIL \longrightarrow LR := LR+1\ ,\ E_0)\ ,$
$7 : (C_0E_0,\ Y[LR+1] = NIL \longrightarrow LA := LR\ ,\ C_0E_0)\ ,$
$8 : (E_0,\ Y[LR+1] = NIL \longrightarrow skip\ ,\ E_0)\ ,$
$9 : (\emptyset,\ MB := NIL\ ,\ \emptyset)\}\ .$

L'invariant I_9 reste respecté (le calcul se termine donc en observant que la deuxième approximation est égale à la première) et on a donc $I_{10} =_{def} I_9$.

On procède exactement de la même manière pour rendre asynchrone la transmission des accusés de réception, au moyen d'un canal modélisé par la nouvelle variable AB (pour "Acknowledgment Buffer"). Le développement du système correspondant et de son invariant est laissé au lecteur.

13.3.4. *Amélioration du parallélisme*

La dernière étape du développement consiste à scinder les transitions en séquences de plusieurs transitions, lorsqu'elles altèrent plusieurs variables. Le développement est relativement fastidieux mais se prête bien à la mécanisation; l'invariant s'adapte sans difficulté. Nous donnons ici immédiatement la version finale (figure 13.2) et son invariant (figure 13.3). On notera que l'invariant final ne diffère guère de l'invariant de la dernière version.

13.3.5. *Conclusion*

13.4. Validation des conditions de vérification

13.4.1. *Introduction*

Nous avons vu que la logique est d'une aide précieuse dans le processus de construction d'un invariant de programme, et donc dans celui de la vérification des programmes. Néanmoins, la technique que nous avons présentée et illustrée plus haut ne donne que des approximations de plus en plus précises; il faut, à un certain moment, "deviner" la limite de ces approximations. Même dans le cas particulier fréquent d'un système à espace d'état fini, détecter la stationnarité de la suite est loin d'être évident. Il faut en effet détecter l'équivalence logique de deux approximations successives F^n et F^{n+1}. Si l'espace d'état est fini, les variables du programme ne prennent qu'un nombre fini de valeurs et il est possible de les coder par des variables propositionnelles. Souvent d'ailleurs, la plupart des variables sont déjà booléennes (comme inP et inQ dans l'algorithme de Peterson) ou équivalentes à de telles variables (comme T qui pouvait prendre les deux valeurs p et q). Le problème à résoudre est donc celui de la validité dans la logique propositionnelle. Il est élémentaire sur

$$
\begin{aligned}
\mathcal{P}_f \;=\; & \{A, B, C, D, E\} \text{ où } \; A = \{A_0, A_1, A_2\},\; B = \{B_0, B_1, B_2\},\\
& C = \{C_0, C_1, C_2\},\; D = \{D_0, D_1, D_2\} \text{ et } \; E = \{E_0, E_1, E_2\};\\
\mathcal{T}_f \;=\; \{\;& 1: \; (A_0,\; HS - LA < W \;\longrightarrow\; \mathit{skip},\; A_1),\\
& 2: \; (A_1,\; HS := HS + 1,\; A_2),\\
& 3: \; (A_2,\; MB := (HS, X[HS]),\; A_0),\\
& 4: \; (B_0,\; LA < r \leq HS \;\longrightarrow\; MB := (r, X[r]),\; B_2),\\
& 5: \; (B_2,\; r := r + 1,\; B_0),\\
& 6: \; (B_0,\; \neg(LA < r \leq HS) \;\longrightarrow\; \mathit{skip},\; B_1),\\
& 7: \; (B_1,\; r := LA + 1,\; B_0),\\
& 8: \; (C_0,\; AB \neq NIL \;\longrightarrow\; (\mathit{ackno}, AB) := (AB, NIL),\; C_1),\\
& 9: \; (C_1,\; \mathit{ackno} \leq LA \;\longrightarrow\; \mathit{skip},\; C_0),\\
& 10: \; (C_1,\; \mathit{ackno} > LA \;\longrightarrow\; \mathit{skip},\; C_2),\\
& 11: \; (C_2,\; LA := \mathit{ackno},\; C_0),\\
& 12: \; (D_0,\; MB \neq NIL \;\longrightarrow\; (m, MB) := (MB, NIL),\; D_1),\\
& 13: \; (D_1,\; Y[m.1] := m.2,\; D_2),\\
& 14: \; (D_2,\; AR[m.1] := 1,\; D_0),\\
& 15: \; (E_0,\; AR[LR + 1] = 1 \;\longrightarrow\; \mathit{skip},\; E_1),\\
& 16: \; (E_0,\; AR[LR + 1] = 0 \;\longrightarrow\; \mathit{skip},\; E_2),\\
& 17: \; (E_1,\; LR := LR + 1,\; E_2),\\
& 18: \; (E_2,\; AB := LR,\; E_0),\\
& 19: \; (\emptyset,\; MB := NIL,\; \emptyset),\\
& 20: \; (\emptyset,\; AB := NIL,\; \emptyset)\}.
\end{aligned}
$$

Figure 13.2 *La version finale $\mathcal{S}_f$*

le plan théorique mais il n'existe pas en pratique de technique suffisamment efficace pour tester la validité de très grandes formules, en toute généralité. Le problème devient encore plus inextricable si l'algorithme à valider comporte quelques variables non bornées ou un paramètre non borné, ce qui est toujours le cas si l'algorithme implique un nombre de processus non fixé à l'avance.

Les perspectives deviennent meilleures si l'on tire parti du fait que les formules à valider ne sont pas des formules quelconques, puisqu'elles sont directement liées à un invariant de programme. Dans cette section nous mettons en évidence certaines propriétés de ces formules, ainsi que certains théorèmes de logique, qui permettent d'obtenir des méthodes raisonnablement efficaces pour la validation des invariants. Nous nous limiterons essentiellement à la partie purement propositionnelle du travail de validation, qui est le plus

$$
\begin{array}{l}
(LA \leq LR \leq HS \leq LA + W) \wedge \\
(at\ A_1 \Rightarrow HS < LA + W) \wedge \\
\\
\forall s\,(Y[s] \in \{X[s], NIL\}) \wedge \\
\forall s\,(1 \leq s \leq LR \Rightarrow Y[s] = X[s]) \wedge \\
\forall s\,(HS < s \Rightarrow (Y[s] = NIL] \wedge AR[s] = 0)) \wedge \\
\\
(MB = NIL \vee \exists s\,[(1 \leq s \leq HS) \wedge MB = (s, X[s])]) \wedge \\
(\neg at\ E_2 \Rightarrow AB \in \{NIL, LR\}) \wedge \\
(at\ E_2 \Rightarrow AB \in \{NIL, LR - 1, LR\}) \wedge \\
\\
\forall s\,([AR[s] = 1 \vee (at\ D_2 \wedge s = m.1)] \equiv Y[s] = X[s]) \wedge \\
\\
(at\ C_1 \Rightarrow ackno \leq LR) \wedge \\
(at\ C_2 \Rightarrow LA < ackno \leq LR) \wedge \\
(at\ D_1 \Rightarrow (m.2 = X[m.1] \wedge m.1 \leq HS)) \wedge \\
(at\ D_2 \Rightarrow (Y[m.1] = X[m.1] \wedge m.1 \leq HS)).
\end{array}
$$

Figure 13.3 *L'invariant final* I_f

facile mais aussi le plus important en pratique.

13.4.2. *Propriétés structurelles des conditions de vérification*

En général, un invariant est un ensemble conjonctif d'assertions, par exemple

$$I =_{def} \bigwedge_{k \in K} a_k .$$

Pour un programme d'une certaine taille, l'ensemble des assertions est grand mais la taille d'une assertion individuelle reste petite; globalement, la taille de l'invariant est au moins du même ordre que celle du programme, et parfois nettement plus grande. L'ensemble des conditions de vérification associées au programme $\mathcal{S}$ et à son invariant I est

$$V =_{def} \bigwedge_{k \in K} \bigwedge_{\tau \in \mathcal{S}} (I \Rightarrow wlp[\tau; a_k]) .$$

Remarque. On utilise ici le fait que l'opérateur *wlp* est $\wedge$-additif; en particulier, $wlp[\tau; (a_k \wedge a_\ell)]$ est équivalent à $wlp[\tau; a_k] \wedge wlp[\tau; a_\ell]$, et $(A \Rightarrow \bigwedge_k B_k)$ est équivalent à $\bigwedge_k (A \Rightarrow B_k)$. En pratique, la taille de V est quadratique en la taille de $\mathcal{S}$.[15]

[15] Ce n'est pas un résultat formel; la taille de $\mathcal{S}$, et plus encore les tailles de I et V dépendent des notations utilisées.

Pour introduire les problèmes liés à la validation systématique des conditions de vérification, nous reconsidérons brièvement un programme d'exclusion mutuelle pour un nombre quelconque n de processus, étudié dans le volume II (cf. II.4.5.1). Les transitions sont

1. $(p_0,\ \neg rcs_p \longrightarrow [\,]\,,\ p_0)$,
2. $(p_0,\ rcs_p \wedge SC = 0 \longrightarrow SC := p\,,\ p_c)$,
3. $(p_0,\ rcs_p \wedge SC \neq 0 \longrightarrow E := E \cup \{p\}\,,\ p_w)$,
4. $(p_c,\ rcs_p \longrightarrow [\,]\,,\ p_c)$,
5. $(p_c q_w,\ \neg rcs_p \wedge q \in E \longrightarrow (SC, E) := (q, E \setminus \{q\})\,,\ p_0 q_c)$,
6. $(p_c,\ \neg rcs_p \wedge E = \emptyset \longrightarrow SC := 0\,,\ p_0)$.

La transition 5 existe pour tout couple (p, q) de processus distincts, les autres existent pour tout processus p; il y a donc en tout $5n + n(n-1)$ transitions. L'invariant est

$$\begin{aligned} I \ =_{def}\ & \forall r\,[at\ r_w \equiv r \in E] \\ \wedge\ & \forall r\,[at\ r_c \equiv SC = r] \\ \wedge\ & (SC = 0 \Rightarrow E = \emptyset)\,. \end{aligned}$$

Il y a donc $2n + 1$ assertions dans cet invariant.

Considérons seulement une transition et une assertion, à savoir

$$\tau =_{def} (p_c q_w,\ \neg rcs_p \wedge q \in E \longrightarrow (CS, E) := (q, E \setminus \{q\})\,,\ p_n q_c)\,,$$
$$a_k =_{def} at\ r_w \equiv r \in E\,,$$

l'assertion $wlp[\tau; a_k]$ se réduit à

$$(13.1) \quad \neg rcs_p \wedge q \in E \Rightarrow (at\ r_w \equiv r \in E \setminus \{q\})\,.$$

Pour démontrer la validité de $(I \Rightarrow wlp[\tau; a_k])$, nous utilisons

- l'une des assertions de l'invariant I, à savoir $(at\ r_w \equiv r \in E)$,
- la convention $q \neq r$ (deux processus de noms distincts sont distincts),
- un résultat de la théorie des ensembles, à savoir $(q \in E \wedge q \neq r) \Rightarrow (r \in E \equiv r \in E \setminus \{q\})$,
- la théorie de l'égalité et la logique propositionnelle.

Une première constatation intéressante est que la forme habituelle d'une condition de vérification telle que $I \Rightarrow wlp[\tau; a_k]$ est

$$(13.2) \quad (a_1 \wedge \cdots \wedge a_m) \Rightarrow (c \Rightarrow a'_k)\,,$$

où a'_k est $wlp[\tau; a_k]\ \tau$.

Un second point est que le nombre m d'assertions dans un invariant augmente rapidement avec la taille du programme considéré. C'est notamment le cas si on vise un codage purement propositionnel de l'invariant. Par exemple, les prédicats de place $at\ p_n$, $at\ p_w$, $at\ p_c$ peuvent être traités comme des propositions ordinaires, à condition que l'on ajoute les assertions exprimant

qu'un seul de ces trois atomes est vrai. Il en va de même pour les égalités du type $CS = p$, qui peuvent être traitées comme des propositions si l'on ajoute des assertions exprimant qu'exactement une seule des égalités de ce type est vraie dans un état.[16] Enfin, des prédicats comme $q \in E$ peuvent aussi être traités comme des atomes, à condition que leurs propriétés spécifiques fassent l'objet d'assertions supplémentaires, comme $\neg(q \in E \wedge E = \emptyset)$. Toutes ces assertions supplémentaires, dites *structurelles* (parce qu'elles modélisent les propriétés des structures de données manipulées) contribuent à alourdir la tâche de validation.

Un troisième point à observer est que, même si le nombre total m d'assertions est élevé, dériver la formule $(c \Rightarrow a'_k)$ au départ de l'ensemble $\{a_i : i = 1, \ldots, m\}$ impliquera l'utilisation de seulement quelques éléments de cet ensemble; pouvoir deviner a priori lesquels constituerait une simplification très substantielle du travail de validation.

13.4.3. *Le problème de la validation*

Le problème consiste à déterminer, pour chaque condition de vérification, si la conclusion $C =_{def} (c \Rightarrow a'_k)$ est conséquence logique du (grand) ensemble d'hypothèses $\mathcal{H} =_{def} \{a_1, \ldots, a_m\}$.

Notre premier objectif sera d'obtenir des règles permettant d'éliminer d'emblée autant d'hypothèses inutiles que possible. Ces règles se baseront sur des caractéristiques syntaxiques des hypothèses et de la conclusion; elles proviennent de [Gribomont 2000], où le lecteur trouvera les démonstrations omises ici, de même que des développements supplémentaires.

13.4.4. *Type, type réduit et polarité*

Nous supposons dans la suite que les seuls connecteurs utilisés dans les formules sont $\neg$, $\wedge$, $\vee$ and $\Rightarrow$ (pas d'équivalence).

Chaque sous-formule (plus précisément, chaque occurrence de sous-formule) possède un *type*, qui est un mot composé des lettres c, d et n. Soit B une formule; son type, comme sous-formule présente dans une formule A, est défini inductivement comme suit :

- Le type de B dans B est λ (mot vide).
- Si le type de B dans A est le mot w, alors
  - son type dans $\neg A$ is nw,
  - son type dans $(A \wedge X)$ ou $(X \wedge A)$ est cw,
  - son type dans $(A \vee X)$, $(X \vee A)$ ou $(X \Rightarrow A)$ est dw,
  - son type dans $(A \Rightarrow X)$ est dnw.

[16] Autrement dit, chaque état attribue une seule valeur à la variable SC.

Remarque. L'implication $(p \Rightarrow q)$ est assimilée à la disjonction $(\neg p \vee q)$; "n", "c" and "d" signifient respectivement *n*égation, *c*onjonction et *d*isjonction.

Soit B_A une occurrence de B dans A, de type w. Le *type réduit* de B_A est obtenu en appliquant à w les règles de réécriture

$$c \longrightarrow ndn\,,\ nn \longrightarrow \lambda\,,$$

correspondant aux lois de De Morgan.

Les notions de type et de type réduit permettent de caractériser syntaxiquement certains concepts sémantiques utiles, à commencer par celui de polarité. La *polarité* d'une occurrence B_A de B dans A est *positive* si le type (ou le type réduit) de B_A contient un nombre pair de "n"; elle est *négative* sinon.
Remarque. Une sous-formule peut avoir plusieurs occurrences dans une formule, de types et de polarités distinctes.

Lemme de polarité. Soient A_t et A_f les formules obtenues en remplaçant une occurrence B_A de B dans A par *true* et *false*, respectivement. Les formules $(A_f \Rightarrow A)$ et $(A \Rightarrow A_t)$ sont des tautologies si B_A est de polarité positive; sinon, $(A_t \Rightarrow A)$ et $(A \Rightarrow A_f)$ sont des tautologies.
Remarque. Plusieurs occurrences de B dans A peuvent être remplacés simultanément, pourvu qu'elles soient toutes de même polarité.

Notation. Si A, B et C sont des formules propositionnelles, alors $A[B/C]$ est la formule obtenue en remplaçant simultanément dans A toutes les occurrences de B par C. (Si B n'est pas une sous-formule de A, alors $A[B/C]$ est simplement A.)

Théorème de polarité. Si la proposition p n'a pas d'occurrence négative dans A ni d'occurrence positive dans B, alors les formules $(A[p/false] \Rightarrow A)$, $(A \Rightarrow A[p/true])$, $(B[p/true] \Rightarrow B)$ et $(B \Rightarrow B[p/false])$ sont des tautologies.

Règle de polarité. Si la proposition p n'a pas d'occurrence négative dans $\mathcal{H}$ ni d'occurrence positive dans C, alors C est conséquence logique de $\mathcal{H}$ si et seulement si $C[p/true]$ est conséquence logique de $\mathcal{H}[p/true]$. Si la proposition p n'a pas d'occurrence positive dans $\mathcal{H}$ ni d'occurrence négative dans C, alors C est conséquence logique de $\mathcal{H}$ si et seulement si $C[p/false]$ est conséquence logique de $\mathcal{H}[p/false]$.

13.4.5. *Rang d'une sous-formule*

Le *rang* d'une occurrence B_A de B dans A est le nombre de "d" précédés (immédiatement ou non) d'un nombre pair de "n" dans le type réduit de B_A. L'occurrence B_A de B dans A est *purement disjonctive* si son type réduit w ne comporte que des lettres d, suivie ou non de la lettre n; elle est *purement conjonctive* if si son type réduit est vide, ou n suivi d'un type réduit purement disjonctif.
Remarque. L'occurrence B_A is purement conjonctive (resp. disjonctive) dans

A si et seulement si elle est purement disjonctive (resp. conjonctive) dans $\neg A$.
Remarque. Une occurrence est purement conjonctive si et seulement si son rang est nul.

Exemples. Soit A la formule $(p \Rightarrow q) \vee \neg(r \wedge (\neg s \Rightarrow \neg(t \Rightarrow u)))$. Les types, types réduits, rangs et polarités des propositions présentes dans A sont données dans le tableau suivant :

occurrence	p	q	r	s	t	u
type	*ddn*	*dd*	*dnc*	*dncdnn*	*dncdndn*	*dncdnd*
type réduit	*ddn*	*dd*	*ddn*	*ddnd*	*ddndndn*	*ddndnd*
purement disjonctive ?	oui	oui	oui	non	non	non
rang	2	2	2	2	3	3
polarité	−	+	−	−	−	+

Lemme du rang. Soit B_A une occurrence purement disjonctive d'une sous-formule B d'une formule A. Si v est une interprétation telle que $v(A) = \mathit{false}$, alors $v(B) = \mathit{false}$ si B_A est de polarité positive, et $v(B) = \mathit{true}$ si B_A est de polarité négative.

Remarque. Un résultat dual existe pour les occurrences purement conjonctives.

Règle du rang. Si p a une occurrence positive, purement conjonctive dans une hypothèse $h \in \mathcal{H}$ ou une occurrence négative, purement disjonctive dans C, alors C est conséquence logique de $\mathcal{H}$ si et seulement si $C[p/\mathit{true}]$ est conséquence logique de $\mathcal{H}[p/\mathit{true}]$. Si p a une occurrence négative, purement conjonctive dans une hypothèse $h \in \mathcal{H}$ ou une occurrence positive, purement disjonctive dans C, alors C est conséquence logique de $\mathcal{H}$ si et seulement si $C[p/\mathit{false}]$ est conséquence logique de $\mathcal{H}[p/\mathit{false}]$.

13.4.6. *Polarité et interpolation*

Un *interpolant* pour les formules H et C est une formule J telle que chaque proposition présente dans J est aussi présente dans H et dans C, et que $(H \Rightarrow J)$ et $(J \Rightarrow C)$ sont des tautologies. Un *interpolant polarisé* pour les formules H et C est un interpolant J tel que toute proposition p présente dans J l'est aussi dans H et dans C, avec la même polarité. Si J existe, $(H \Rightarrow C)$ est une tautologie; la réciproque est le théorème d'interpolation de Craig.

Théorème d'interpolation. Si $(H \Rightarrow C)$ est une tautologie, alors H et C admettent un interpolant.

Remarque. Cette proposition et la suivante sont des cas particuliers de résultats classiques de la théorie des modèles en logique prédicative (voir par

exemple [Chang et Keisler 73], pp. 84-90). Une preuve constructive des cas propositionnels est donnée dans [Gribomont 2000].
Théorème d'interpolation polarisée. Si $(H \Rightarrow C)$ est une tautologie, alors H et C admettent un interpolant polarisé.

13.4.7. *Pertinence et règle d'élimination*

Définition. Soit Π un ensemble de variables propositionnelles et H et C des formules sur le lexique Π. La formule H est *pertinente pour C par rapport à Π* si une interprétation v sur Π existe telle que $v(H) = v(C) = \mathit{false}$.
Remarques. Cette définition ne capture pas exactement la notion intuitive de pertinence mais une relation plus large. Cependant, le théorème suivant montre que les hypothèses non pertinentes sont en fait inutiles (et donc non pertinentes au sens intuitif). Observons aussi que la pertinence est une relation symétrique. Deux variables propositionnelles sont toujours mutuellement pertinentes. En pratique, le lexique de référence Π n'est pas mentionné explicitement.
Théorème de la pertinence. Si h_1 n'est pas pertinent pour C par rapport à Π, alors les formules

$$\phi_1 \quad =_{def} \quad (h_1 \wedge h_2 \wedge \cdots \wedge h_n) \Rightarrow C,$$
$$\phi_2 \quad =_{def} \quad (h_2 \wedge \cdots \wedge h_n) \Rightarrow C,$$

sont logiquement équivalentes, quelles que soient les formules $h_2, \ldots, h_n$. Réciproquement, si h_1 est pertinent pour C par rapport à Π, alors des formules $h_2, \ldots, h_n$ existent telles que ϕ_1 soit valide et ϕ_2 soit non valide.
Règle d'élimination. Soit A une sous-formule commune à H et C, et A_H et A_C les occurrences correspondantes. Si A_H et A_C sont puremenr disjonctives et de polarités opposées, alors H est non pertinent pour C.
Exemple. Soient $H =_{def} \alpha \vee (\beta \Rightarrow \gamma)$ et $C =_{def} \alpha \vee (\gamma \Rightarrow \delta)$. La sous-formule γ est purement disjonctive dans H et dans C, positive dans H et négative dans C. La formule H n'est donc pas pertinente pour C.
Remarque. La règle d'élimination est utile parce qu'elle est facile à appliquer et permet d'écarter rapidement un certain nombre d'hypothèses sans intérêt.

13.4.8. *La règle de connectivité*

Soient $\mathcal{H}$ un ensemble consistant d'hypothèses et C une conclusion. Une *chaîne polarisée* est une suite non vide $(\phi_0, \ldots, \phi_n)$ telle que

- ϕ_0 est $\neg C$;
- $\phi_i \in \mathcal{H}$, pour tous $i = 1, \ldots, n$;
- ϕ_i et ϕ_{i-1} comportent au moins une proposition en commun, pour tous $i = 1, \ldots, n$.
- ϕ_i est pertinente pour $\neg\phi_{i-1}$, pour tous $i = 1, \ldots, n$.

L'ensemble $\mathcal{H}$ et la formule C étant donnés, on note $\mathcal{H}_C$ l'ensemble des hypothèses qui apparaissent dans au moins une chaîne polarisée.

Théorème de la connectivité polarisée. La conclusion C est une conséquence logique de l'ensemble $\mathcal{H}$ des hypothèses si et seulement si cette conclusion est conséquence logique du sous-ensemble $\mathcal{H}_C \subset \mathcal{H}$.

13.4.9. *Un exemple*

Nous montrons ici comment la validation d'une condition de vérification peut être grandement facilitée par l'emploi des règles que nous venons d'introduire. On utilise le lexique propositionnel suivant :

$$\Pi =_{def} \{a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2, e_0, e_1, e_2, f_0, f_1, f_2\}.$$

Les hypothèses de la condition de vérification sont

$$\begin{aligned}
H_1 &: c_1 \\
H_2 &: a_2 \vee (c_1 \Rightarrow e_1) \\
H_3 &: b_0 \Rightarrow \neg(a_2 \vee e_0) \\
H_4 &: e_1 \Rightarrow \neg(a_2 \wedge c_2) \\
H_5 &: e_1 \Rightarrow (a_0 \Rightarrow d_0) \\
H_6 &: (a_0 \wedge d_0) \Rightarrow ((c_0 \wedge d_2) \Rightarrow f_0) \\
H_7 &: (f_1 \wedge f_0) \Rightarrow ((a_0 \wedge a_2) \vee (c_0 \wedge e_0)) \\
H_8 &: f_2 \Rightarrow (c_0 \vee e_2) \\
H_9 &: ((a_0 \vee c_0) \wedge f_0) \Rightarrow (f_1 \vee (e_0 \wedge f_2)) \\
H_{10} &: (\neg c_2 \vee f_0) \Rightarrow (e_2 \vee (b_2 \wedge \neg f_1)) \\
H_{11} &: ((d_0 \vee e_0) \Rightarrow ((b_0 \Rightarrow f_0) \vee (f_0 \Rightarrow c_2))) \\
H_{12} &: \neg c_2 \Rightarrow (e_2 \vee (b_2 \wedge \neg f_1)) \\
H_{13} &: (a_2 \wedge f_0) \Rightarrow (b_0 \vee (e_0 \wedge e_1)) \\
H_{14} &: ((a_1 \wedge c_1) \vee (b_1 \wedge a_2)) \Rightarrow a_0 \\
H_{15} &: ((d_0 \vee a_2) \wedge \neg b_0) \Rightarrow (c_2 \vee (c_0 \Rightarrow f_0)) \\
H_{16} &: ((a_1 \vee c_1) \wedge d_0) \Rightarrow (b_0 \vee (e_0 \wedge a_1)) \\
H_{17} &: b_1 \Rightarrow a_1 \\
H_{18} &: (d_2 \vee (a_1 \Rightarrow f_1)) \Rightarrow ((b_1 \wedge c_1) \vee f_2) \\
H_{19} &: (a_2 \vee c_1) \Rightarrow (b_1 \Rightarrow b_2) \\
H_{20} &: (d_1 \vee f_1) \Rightarrow (a_1 \wedge c_2) \\
H_{21} &: (c_2 \wedge b_2) \Rightarrow (d_1 \vee d_2)
\end{aligned}$$

La conclusion est

$$C : (a_0 \wedge b_0) \Rightarrow ((c_0 \vee c_1) \Rightarrow ((d_0 \vee e_0) \vee (c_2 \Rightarrow e_0)))$$

On peut simplifier le problème de validation comme suit :

1. Elimination de H_{11} (la règle d'élimination s'applique à la sous-formule $(d_0 \vee e_0)$).
2. Elimination de H_{15} (la règle d'élimination s'applique à la proposition b_0).
3. Elimination de H_6 et H_{16} (la règle d'élimination s'applique à la proposition d_0).
4. La proposition f_0 n'a que des occurrences négatives dans les hypothèses restantes, et pas d'occurrence du tout dans la conclusion; f_0 peut être uniformément remplacée par *false* (règle de polarité).
5. Cela permet l'élimination subséquente de H_7, H_9 and H_{13}, devenues triviales; de plus, H_{10} se simplifie et se confond avec H_{12}.
6. Le théorème de connectivité polarisée permet enfin l'élimination de H_8, H_{10}, H_{12}, H_{17}, H_{18}, H_{19}, H_{20} et H_{21}.

Les hypothèses restantes sont

$$H_1 : c_1 \qquad H_4 : e_1 \Rightarrow \neg(a_2 \wedge c_2)$$
$$H_2 : a_2 \vee (c_1 \Rightarrow e_1) \qquad H_5 : e_1 \Rightarrow (a_0 \Rightarrow d_0)$$
$$H_3 : b_0 \Rightarrow \neg(a_2 \vee e_0) \qquad H_{14} : ((a_1 \wedge c_1) \vee (b_1 \wedge a_2)) \Rightarrow a_0$$

La proposition c_1 a une occurrence purement conjonctive dans H_1 et est remplacée par *true* (règle du rang); de même, a_0 et b_0 ont des occurrences purement disjonctives dans C et sont remplacées par *true*. Les hypothèses H_1 et H_{14} disparaissent, et l'ensemble des hypothèses restantes se réduit à

$$H_2 : a_2 \vee e_1 \qquad H_4 : e_1 \Rightarrow \neg(a_2 \wedge c_2)$$
$$H_3 : \neg(a_2 \vee e_0) \qquad H_5 : e_1 \Rightarrow d_0 ;$$

La conclusion simplifiée est

$$d_0 \vee e_0 \vee (c_2 \Rightarrow e_0) .$$

La validation est maintenant immédiate.

Bibliographie

[Ackermann 56] W. Ackermann, Begründung einer Strengen Implikation, *Journal of Symbolic Logic*, vol. 21, p. 113-128, 1956.

[Alimi et al. 99] J. Alimi, G. Dowek et L. Rolland, in N. Farouki (éditeur), *Quand la Science a dit...c'est impossible*, Le Pommier-Fayard, Paris 1999.

[Anderson and Belnap 62] A. Anderson and N. Belnap, The pure calculus of entailment, *The Journal of Symbolic Logic*, vol. 27, p. 19-52, 1962.

[Anderson and Belnap 68] A. Anderson and N. Belnap, Entailment, in G. Iseminger (éditeur), *Logic and Philosophy*, p. 76-110, Appelton-Century-Crofts, New York, 1968.

[Anderson and Belnap 75] A. Anderson and N. Belnap, *Entailment. The Logic of Relevance and Necessity*, vol. 1, Princeton University Press, Princeton, 1975.

[Anderson and Johnstone 62] A. Anderson and H. Johnstone, *Natural Deduction*, Wadsworth, Belmont, 1962.

[Aristote 59] Aristote, *Organon I, Catégories II, De l'Interprétation*, traduction de J. Tricot, Vrin, Paris, 1959.

[Aristote 47] Aristote, *Organon III, Les Premiers Analytiques*, traduction de J. Tricot, Vrin, Paris, 1947.

[Aristote 62] Aristote, *Organon IV, Les Seconds Analytiques*, traduction de J. Tricot, Vrin, Paris, 1962.

[Bailhache 91] P. Bailhache, *Essai de Logique Déontique*, Vrin, Paris, 1991.

[Barcan 46] R. Barcan, A functional calculus of first order based on strict

implication, *The Journal of Symbolic Logic*, vol. 11, no 1, March, p. 1-16, 1946.

[Barendregt 80] H. Barendregt, *The Lambda Calculus, its Syntax and Semantics*, North-Holland, Amsterdam, 1980.

[Barreau 84] H. Barreau, Diodore Cronos, in D. Huisman (éditeur), *Dictionnaire des Philosophes*, Presses Universitaires de France, p. 754-759, 1984.

[Basin et al. 97] D. Basin, S. Matthews and L. Vigano, Labelled quantified modal logic, in G. Brewka et al. (éditeurs), *KI-97 : Advances in Artificial Intelligence*, p. 171-182, Springer-Verlag, Berlin, 1997.

[Basin et al. 98] D. Basin, S. Matthews and L. Vigano, Labelled modal logics : quantifiers, *Journal of Logic, Language and Information*, vol. 7, p. 237-263, 1998.

[Batens 95] D. Batens, Blocks. The clue to dynamic aspects of logic, *Logique et Analyse*, juin-septembre, no 150-152, décembre, p. 285-327, 1995.

[Bayart 58] A. Bayart, La correction de la logique modale du premier et du second ordre $S5$, *Logique et Analyse I*, no 1, janvier, p. 28-45, 1958.

[Bayart 59] A. Bayart, Quasi-adéquation de la logique modale du second ordre $S5$ et adéquation de la logique modale du premier ordre $S5$, *Logique et Analyse II*, no 6-7, avril, p. 28-45, 1959.

[Bell and Slomson 69] J. Bell and A. Slomson, *Models and Ultraproducts*, North Holland, Amsterdam, 1969.

[Ben Ari 93] M. Ben Ari, *Mathematical Logic for Computer Sciences*, Prentice Hall, New York, 1993.

[Bencivenga 86] E. Bencivenga, Free logic, in D. Gabbay and F. Guenthner (éditeurs), *Handbook of Philosophical Logic, vol. 3, Alternatives in Classical Logic*, p. 373-426, 1986.

[Binot 90] J.-L. Binot, Traitement de la langue naturelle, logique et programmation, in A. Thayse (éditeur), *Approche Logique de l'Intelligence Artificielle, tome 3 : Du Traitement de la Langue à la Logique des Systèmes Experts*, Dunod, Paris, 1990.

[Blackburn et al.] P. Blackburn, M. de Rijke and I. Venema, *Modal Logic*, à paraître.

[Bläsius und Bürkhert 92] K. Bläsius und H. Bürkhert, *Automatisierung des Logischen Denkens*, 2ième édition, Oldenburg, Munich, 1992.

[Bloesch 93a] A. Bloesch, *Signed Tableaux. A Basis for Automated Proving in non Classical Logics*, Ph. D. Thesis, University of Queensland, Australia, 1993.

[Bloesch 93b] A. Bloesch, A tableau style proof system for two paraconsistent logics, *Notre Dame Journal of Formal Logic*, vol. 34, p. 295-301, 1993.

[Bratko 90] I. Bratko, *Prolog Programming for Artificial Intelligence*, Addison-Wesley, Reading, Mass., 2ième édition, 1990.

[Carnap 36] R. Carnap, *The Logical Syntax of Language*, Routledge and Kegan, Londres, 1936.

[Carnap 46] R. Carnap, Modalities and quantification, *The Journal of Symbolic Logic*, vol. 11, no 2, June, p. 33-64, 1946.

[Carnap 47] R. Carnap, *Meaning and Necessity : A Study in Semantics and Modal Logic*, The University of Chicago Press, Chicago, 1947.

[Carnap 97] R. Carnap, *Signification et Nécessité*, traduction et présentation de F. Rivenc et Ph. de Rouilhan, Gallimard, Paris 1997.

[Castilho et al. 97] M. Castilho, L. Fariñas del Cerro, O. Gasquet and A. Herzig, Modal tableaux with propagation rules and structural rules, *Fundamenta Informaticae*, vol. 32, p. 291-297, 1997.

[Ceri et al. 90] S. Ceri, G. Gottlob and L. Tanca, *Logic Programming and Databases*, Springer-Verlag, Berlin, 1990.

[Chagrov and Zakharyaschev 97] A. Chagrov and M. Zakharyaschev, *Modal Logic*, Clarendon Press, Oxford, 1997.

[Chang and Keisler 73] C. Chang and H. Keisler, *Model Theory*, North-Holland, Amsterdam, 1973.

[Chellas 80] B. Chellas, *Modal Logic*, Cambridge University Press, Cambridge, 1980.

[Church 51] A. Church, The weak theory of implication. Kontrolliertes Denken, *Festgabe zum 60 Geburtstag van Prof. W. Britzelmayr*, Munich 1951.

[Clifford 90] J. Clifford, *Formal Semantics and Pragmatics for Natural Language Querying*, Cambridge Tracts in Theoretical Computer Science, vol. 8, 1990.

[Cocchiarella 66] N. Cocchiarella, *Tense Logic : A Study of Temporal Reference*, Ph. D. Thesis, U.C.L.A., 1966.

[Cocchiarella 84] N. Cocchiarella, Philosophical perspectives on quantification in tense and modal logic, in D. Gabbay and F. Guenthner (éditeurs), *Handbook of Philosophical Logic, vol. 2, Extensions in Classical Logic*, Reidel, Dordrecht, p. 309-353, 1984.

[Codd 70] E. Codd, A relational model of data for large shared data banks, *Communications of the ACM*, vol. 13, p. 377-387, 1970.

[Chomsky 59] N. Chomsky, On certain formal properties of grammars, *Information and Control*, vol. 2, p. 137-167, 1959.

[Colmerauer 78] A. Colmerauer, Metamorphosis grammars, in L. Bole (éditeur), *Natural Language Communication with Computers*, p. 139-169, *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, vol. 63, 1978.

[Colmerauer et al. 83] A. Colmerauer, H. Kanoui et M. Van Caneghem, Prolog, bases théoriques et développements actuels, *T.S.I.*, vol. 2, no 4, 1983.

[Copeland 96] J. Copeland (éditeur), *Logic and Reality, Essays on the Legacy of Arthur Prior*, Clarendon Press, Oxford, p. 1-40, 1996.

[Crabbé 86] M. Crabbé, Le calcul lambda, *Cahiers du Centre de Logique*, no 6 : *Logique et Informatique*, Cabay, Louvain-la-Neuve, p. 63-86, 1986.

[Crocco 96] G. Crocco, *Fondements Logiques du Raisonnement Contextuel. Une Etude sur les Logiques des Conditionnels*, Unipress, Padova, 1996.

[D'Agostino and Mondadori 94] M. D'Agostino and M. Mondadori, The taming of the cut. Classical refutations with analytic cut, *J. Logic Computat.*, vol. 4, p. 285-319, 1994.

[Davio et al. 78] M. Davio, J.-P. Deschamps and A. Thayse, *Discrete and Switching Functions*, McGraw-Hill, New York 1978.

[Dean et al. 95] T. Dean, J. Allen et Y. Aloimonos, *Artificial Intelligence, Theory and Practice*, Benjamin-Cummings, Redwood, California, 1995.

[Dekker 89] P. Dekker, Dynamic interpretation, negation and flexibility, *First European Summer School on Natural Language Processing, Logic and Knowledge Representation*, Groningen, June 5-17, 1989.

[Dekker 90] P. Dekker, Dynamic interpretation, flexibility and monotonicity, in M. Stokhof and L. Torenvliet (éditeurs), *Proceedings of the Seventh Amsterdam Colloquium, 1989*, Amsterdam, 1990.

[Delahaye 86] J. Delahaye, *Outils Logiques pour l'Intelligence Artificielle*, Eyrolles, Paris, 1986.

[de Rijke 93] M. de Rijke, *Extending Modal Logic*, Dissertation, Institute for Logic, Language and Computation, University of Amsterdam, 1993.

[De Swart 94] H. De Swart, *Mathematics, Language, Computer Science*, Peter Lang, Bern, 1994.

[Doets 94] K. Doets, *From Logic to Logic Programming*, MIT Press, Cambridge, Mass., 1994.

[Došen 92] D. Došen, The first axiomatization of relevant logic, *Journal of Philosophical Logic*, vol. 21, p. 339-356, 1992.

[Dowty et al. 81] D. Dowty, R. Wall and S. Peters, *Introduction to Montague Semantics*, Reidel, Dordrecht 1981.

[Enderton 72] H. Enderton, *A Mathematical Introduction to Logic*, Academic Press, New York and London, 1972.

[Engel] P. Engel, Modalities (Logique), *Encyclopedia Universalis*, Paris, p. 509-511.

[Fagin et al. 95] R. Fagin, J. Halpern, Y. Moses, M. Vardi, *Reasoning about Knowledge*, The MIT Press, Cambridge, Mass., 1995.

[Farreny et Ghallab 87] H. Farreny et M. Ghallab, *Eléments d'Intelligence Artificielle*, Hermès, Paris, 1987.

[Feys 50] R. Feys, Les systèmes formalisés des modalités aristotéliciennes, *Revue Philosophique de Louvain*, p. 478-509, 1950.

[Fine and Schurz 96] K. Fine and G. Schurz, Transfer theorems for multimodal logics, in J. Copeland (éditeur), *Logic and Reality*, Clarendon Press, Oxford, 1996.

[Fitch 52] F. Fitch, *Symbolic Logic. An Introduction*, The Ronald Press, New York, 1952.

[Fitting 83] M. Fitting, *Proof Methods for Modal and Intuitionistic Logics*, Reidel, Dordrecht, 1983.

[Fitting 93] M. Fitting, Basic modal logic, in D. Gabbay et al. (éditeurs), *Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 1 : Elements of Classical Logic*, Clarendon Press, Oxford, p. 318-448, 1993.

[Fitting and Mendelsohn 98] M. Fitting and R. Mendelsohn, *First-Order Modal Logic*, Kluwer, Dordrecht, 1998.

[Forbes 94] G. Forbes, *Modern Logic. A Text in Elementary Symbolic Logic*, Oxford University Press, Oxford, 1994.

[Frege 52, 71] G. Frege, Über Sinn und Bedentung, *Zeitschrift für Philosophie und Philosophische Kritik*, p. 25-50, vol. 100, 1893; traduit dans P. Geach and M. Black (éditeurs), *Translation from the Philosophical Writings of Gottlob Frege*, p. 56-78, Basil Blackwell, Oxford, 1952. Traduction française et introduction par C. Imbert, *Ecrits Logiques et Philosophiques*, p. 102-126, Le Seuil, Paris, 1971.

[Gabbay 96] D. Gabbay, *Labelled Deductive Systems*, Clarendon Press, Oxford, 1996.

[Gabbay and Guenthner 83] D. Gabbay and F. Guenthner (éditeurs), *Handbook of Philosophical Logic; vol. 1 : Elements of Classical Logic*, Reidel, Dordrecht, 1983.

[Gabbay and Guenthner 84] D. Gabbay and F. Guenthner (éditeurs), *Handbook of Philosophical Logic; vol. 2 : Extensions of Classical Logic*, Reidel, Dordrecht, 1984.

[Gabbay and Guenthner 86] D. Gabbay and F. Guenthner (éditeurs), *Handbook of Philosophical Logic; vol. 3 : Alternatives to Classical Logic*, Reidel, Dordrecht, 1986.

[Gabbay and Guenthner 89] D. Gabbay and F. Guenthner (éditeurs), *Handbook of Philosophical Logic; vol. 4 : Topics in the Philosophy of Language*, Reidel, Dordrecht, 1989.

[Gabbay et al. 95] D. Gabbay, C. Hogger and J. Robinson (éditeurs), *Handbook of Logic in Artificial Intelligence and Logic Programming; vol. 4 : Epistemic*

and Temporal Reasoning, Oxford Science Publications, 1995.

[Gallier 86] J. Gallier, *Logic for Computer Science*, Harper and Row, New York, 1986.

[Galton 95] A. Galton, *Time and Change for AI*; in D. Gabbay, C. Hogger and J. Robinson (éditeurs), *Handbook of Logic in Artificial Intelligence and Logic Programming; vol. 4 : Epistemic and Temporal Reasoning*, Oxford Science Publications, 1995.

[Gardies 75] J. Gardies, *La Logique du Temps*,Presses Universitaires de France, Paris, 1975.

[Gardies 79] J. Gardies, *Essai sur la Logique des Modalités*, Presses Universitaires de France, Paris, 1979.

[Gardner] M. Gardner, *Martin Gardner's column in Scientific American*, voir `http://www.scientificamerican.com/` et `http://www.amazon.com/`

[Garson 84] J. Garson, *Quantification in Modal Logic*; in D. Gabbay and F. Guenthner (éditeurs), *Handbook of Philosophical Logic; vol. 2: Extensions of Classical Logic*, Reidel, Dordrecht, 1984.

[Genesereth and Nilsson 87] M. Genesereth and N. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufman, Los Altos, 1987.

[Ginsberg 93] M. Ginsberg, *Essentials of Artificial Intelligence*, Morgan Kaufmann, 1993.

[Gochet 75] P. Gochet, La logique du sens, in *Les Langages, le Sens et l'Histoire*, Presses Universitaires de Lille, p. 225-240, Lille, 1975.

[Gochet 82a] P. Gochet, La Sémantique récursive de Davidson et de Montague, in M. Loi (éditeur), *Penser les Mathématiques*, p. 73-87, Le Seuil, Paris, 1982.

[Gochet 82b] P. Gochet, L'originalité de la sémantique de Montague, *Etudes Philosophiques*, p. 149-175, 1982.

[Gochet 83] P. Gochet, La sémantique des situations, in F. Nef (éditeur), *Histoire, Epistémologie, Langage, tome 5, fascicule 2 : La Sémantique Logique*, pp. 195-212, 1983.

[Gochet 86] P. Gochet, *Ascent to Truth*, Philosophia Verlag, Berlin, 1986.

[Gochet 95] P. Gochet, Problèmes de logique de la connaissance, in J. Dubucs et F. Lepage, *Méthodes pour les Sciences Cognitives*, p. 281-294, Hermès, Paris 1995.

[Gochet 97] P. Gochet, Le syllogisme pratique d'Aristote, in J. Denooz et A. Motte (éditeurs), *Mélanges en l'honneur de C. Rutten*, LASLA, pp. 117-129, Liège, 1997.

[Gochet 01] P. Gochet, Logic, existence and ontology, in D. Jacquette (éditeur), *A Companion to Philosophical Logic*, Blackwell, Oxford, à paraître, 2001.

[Gochet et al. 95] P. Gochet, P. Gribomont and D. Rossetto, Algorithms for relevant logic, *Logique et Analyse*, juin-septembre-décembre, p. 150-152, 329-346, 1995.

[Gochet and Gillet 99] P. Gochet and E. Gillet, Quantified modal logic. Dynamic semantics and $S5$, *Dialectica*, p. 243-250, 1999.

[Gochet et Gribomont 91] P. Gochet et P. Gribomont, *Logique, volume 1*, Hermès, Paris, 1991.

[Gochet et Gribomont 94] P. Gochet et P. Gribomont, *Logique, volume 2*, Hermès, Paris, 1994.

[Goldblatt 87] R. Goldblatt, *Logics of Time and Computation*, Lecture Notes no 7, Center for the Study of Language and Information, 1987.

[Goldblatt 92] R. Goldblatt, *Logics of Time and Computation*, Lecture Notes no 7, Center for the Study of Language and Information, revised edition, 1992.

[Goré 92] R. Goré, *Cut-free Sequent and Tableau Systems for Propositional Normal Modal Logics*, Technical Report 257, University of Cambridge, Cambridge, 1992.

[Goré 95] R. Goré, *Tableau Methods for Modal and Temporal Logics*, Technical Report TR-ARP-15-95, R.S.S.S., Australian National University, 1995.

[Gribomont 85] P. Gribomont, *Synthesis of Parallel Programs Invariants*, *Lecture Notes in Computer Science*, vol. 186, p. 325-338, Springer-Verlag, Berlin, 1985.

[Gribomont 00] P. Gribomont, Simplification of Boolean verification conditions, à paraître, *Theoretical Computer Science*, 2000.

[Granger 95] G. Granger, *Le probable, le possible et le virtuel*, Editions Odile Jacob, Paris 1995.

[Groenendijk and M. Stokhof 89] J. Groenendijk and M. Stokhof, Dynamic Predicate Logic; towards a compositional, non-representational semantics of discourse, *First International Summer School on Natural Language Processing, Logic and Knowledge Representation*, Groningen, 1989; aussi dans *Linguistics and Philosophy*, vol. 14, p. 39–100, 1991.

[Groenendijk and Stokhof 90] J. Groenendijk and M. Stokhof, Dynamic Montague grammar, *Proceedings of the Second Symposium on Logic and Language*, Hajduszboszlo, Hongrie, 1990.

[Guenthner and Guenthner-Reutter 78] F. Guenthner and M. Guenthner-Reutter, *Meaning and Translation (Philosophical and Linguistic Approaches)*, New York University Press, New York, 1978.

[Guillaume 58a, b, c] M. Guillaume, Rapports entre calculs propositionnels modaux et topologie impliqués par certaines extensions de la méthode des tableaux, Systèmes de Feys-von Wright, *Comptes rendus de l'Académie des*

Sciences, vol. 246, p. 1140-1142; Système *S*4 de Lewis, *ibidem*, p. 2207-2210; Système *S*5 de Lewis, *ibidem*, vol. 247, p. 1282-1284; Paris, 1958.

[Haack 74, 96] S. Haack, *Deviant Logic*, Cambridge University Press; Cambridge 1974, revised edition, Chicago University Press, Chicago, 1996.

[Halpern 95] J. Halpern, Reasoning about Knowledge : A survey; in D. Gabbay, C. Hogger and J. Robinson (éditeurs), *Handbook of Logic in Artificial Intelligence and Logic Programming; vol. 4 : Epistemic and Temporal Reasoning*, Oxford Science Publications, 1995.

[Halpern et al. 95] J. Halpern, Y. Moses and M. Vardi, *Reasoning About Knowledge*, The MIT Press, Cambridge Mass., 1995.

[Harel 84] D. Harel, Dynamic Logic, in D. Gabbay and F. Guenthner (éditeurs), *Handbook of Philosophical Logic, vol. 2, Extensions of Classical Logic*, vol. 2, p. 497-604, Reidel, Dordrecht, 1984.

[Haton et al. 91] J.-P. Haton et al., *Le Raisonnement en Intelligence Artificielle*, InterEditions, Paris, 1991.

[Hendriks 89] H. Hendriks, Flexible Montague grammar, *First European Summer School on Natural Language Processing, Logic and Knowledge Representation*, Groningen, 1989.

[Hilpinen 81] R. Hilpinen (éditeur), *New Studies in Deontic Logic, Norms, Actions and the Foundations of Ethics*, Reidel, Dordrecht, 1981.

[Hintikka 57] J. Hintikka, Quantifiers in deontic logic, *Societas Scientiarum Fennica Commentationes Humanorum Litterarum*, vol. 23, 1957.

[Hintikka 62] J. Hintikka, *Knowledge and Belief*, Cornell Univ. Press, Ithaca 1962.

[Hintikka 69] J. Hintikka, *Models for Modalities*, Reidel, Dordrecht, 1969.

[Hintikka 70] J. Hintikka, Existential and uniqueness presuppositions, in Lambert, (éditeur), *Philosophical Problems in Logic*, Reidel, Amsterdam, 1970.

[Hintikka 73] J. Hintikka, Grammar and logic, some borderline problems, in J. Hintikka, J. Moravcsik and P. Suppes (éditeurs), *Approaches to Natural Languages*, Reidel, Dordrecht, p. 197-214, 1973.

[Hintikka 81] J. Hintikka, Standard vs. Nonstandard logics : higher-order, modal and first-order logics, in E. Agazzi, *Modern Logics; a Survey*, p. 283-296, Reidel, Dordrecht, 1981.

[Hintikka 89] J. Hintikka, *L'Intentionnalité et les Mondes Possibles*, traduction et présentation de N. Lavand, Presses universitaires de Lille, 1989.

[Hintikka and Sandu 95] J. Hintikka and G. Sandu, The fallacies of the new theory of reference, *Synthese*, p. 245-283, 1995.

[Hopcroft and Ullman 79] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, Mass., 1979.

[Hughes and Cresswell 84] G. Hughes and M. Cresswell, *A Companion to Modal Logic*, Methuen, London, 1984.

[Hughes and Cresswell 96] G. Hughes and M. Cresswell, *A New Introduction to Modal Logic*, Methuen, London, 1996.

[Husserl 59] E. Husserl, *Recherches Logiques; Tome Premier*, traduction H. Elie, Presses Universitaires de France, Paris, 1959.

[Indrzejczak 99] A. Indrzejczak, A survey of natural deduction systems for modal logics, *Logica Trianguli*, vol. 3, p. 55-83, 1999.

[Iseminger] G. Iseminger (éditeur), *Logic and Philosophy*, Appelton-Century-Crofts, New York.

[Jackson et al. 89] P. Jackson, H Reichgelt and F. van Harmelen, *Logic-Based Knowledge Representation*, MIT Press Series in Logic Programming, Cambridge, Mass., 1989.

[Jansana 90] R. Jansana, *Una introducción a la lógica modal*, tecnos, Madrid, 1990.

[Jansana 95] R. Jansana, *Introduction to Modal Logic*, ESSLLI 7th., Barcelona, 1995.

[Janssen 86] T. Janssen, *Foundations and Applications of Montague Grammar, Part I : Philosophy, Framework, Computer Science*, CWI Tract 19, Amsterdam, Center for Mathematics and Computer Science, 1986.

[Jayez 88] J. Jayez, *L'Inférence en Langue Naturelle*, Hermès, Paris, 1988.

[Kalinowski 96] G. Kalinowski, *La Logique Déductive, Essai de Présentation aux Juristes*, Presses Universitaires de France, Paris, 1996.

[Kamp 84] J. Kamp, A Theory of truth and semantic representation, in Groenendijk et al., (éditeurs), *Truth, Interpretation and Information*, p. 1-41, Foris Publications, Dordrecht, 1984.

[Kanger 57] S. Kanger, *Provability in Logic*, Almqvist and Wiksell, Stockholm, 1957.

[Keenan and Faltz 85] E. Keenan and L. Faltz, *Boolean Semantics for Natural Language*, Reidel, Dordrecht, 1985.

[Kleene 71] S. Kleene, *Logique Mathématique*, (trad. de *Mathematical Logic* 1967), Armand Colin, Paris, 1971.

[Kneale and Kneale 62] W. Kneale and M. Kneale, *The Development of Logic*, Clarendon Press, Oxford 1962.

[Knuuttila 81] S. Knuuttila, The Emergence of Deontic Logic in the Fourteenth Century, in R. Hilpinen (éditeur), *New Studies in Deontic Logic, Norms, Actions and the Foundations of Ethics*, p. 225-248, Reidel, Dordrecht, 1981.

[Konolige 86] K. Konolige, *A Deduction Model of Belief*, Research Notes in Artificial Intelligence, Pitman, London, 1986.

[Kripke 59] S. Kripke, A completeness theorem in modal logic, *The Journal of Symbolic Logic*, vol. 24, no 1, p. 1-14, 1959.

[Kripke 63] S. Kripke, Semantic analysis of modal logic : normal propositional calculi, *Zeit. Math. Logik Grund. Math.*, vol. 9, p. 67-96, 1963.

[Kripke 71] S. Kripke, Identity and necessity, in M. K. Munitz (éditeur), *Identity and Individuation*, New York University Press, New York, p. 135-164, 1971.

[Kripke 72] S. Kripke, *La Logique des Noms Propres*; traduction de P. Jacob et F. Récanati, Paris, Editions de Minuit, 1982.

[Kuiper] R. Kuiper and W. Penczek, Verification by hand using linear time temporal logic, in H.C.M. De Swart (éditeur), *Mathematics, Language, Computer Science and Philosophy, vol. 2, Logic and Computer Science*, p. 229-251,

[Lambert 70] K. Lambert, (éditeur), *Philosophical Problems in Logic*, Reidel, Amsterdam, 1970.

[Laurière 87] J. Laurière, *Intelligence Artificielle, Résolution de Problèmes par l'Homme et la Machine*, Eyrolles, Paris, 1987.

[Lavand 89] N. Lavand, Introduction, voir J. Hintikka, *L'Intentionnalité et les Mondes Possibles*, p. 10-34, 1989.

[Leibniz 03] G. Leibniz, Vérités nécessaires et contingentes, in L. Couturat (éditeur), *Opuscules et Fragments Inédits de Leibniz*, p. 16-24, Alcan, Paris, 1903.

[Lemmon 57] E. Lemmon, Is there only one correct system of modal logic, *The Aristotelian Society*, Supplementary volume, p. 23-40, Harrison, London, 1957.

[Lewis 12] C. Lewis, Implication and the algebra of logic, *Mind*, vol. 21, 1912.

[Lewis 18] C. Lewis, *A Survey of Symbolic Logic*, University of California Press, Berkeley, 1918.

[Lewis 73] C. Lewis, *Counterfactuals*, Harvard University Press, Cambridge, 1973.

[Lewis and Langford 32, 59] C. Lewis and C. Langford, *Symbolic Logic*, 1932; Dover Publications (2e édition), 1959.

[Lewis 18] D. Lewis, *A Survey of Symbolic Logic*, University of California, Berkeley, 1918.

[Linski 72] L. Linski (éditeur), *Reference and Modality*, p. 63-72, Oxford University Press, 1972.

[Lloyd 87] J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag (2e édition), Berlin, 1987.

[Lucas 85] T. Lucas, La logique modale, *Revue Philosophique de Louvain*, no 60, p. 595-598, 1985.

[Luger and Stubblefield 98] G. Luger and W. Stubblefield, *Artificial Intelligence*, Addison-Wesley, (3e édition), Reading, Mass., 1998.

[Łukasiewicz 30] W. Łukasiewicz, Many-valued systems of propositional logic, in S. McCall (éditeur), *Polish Logic*, Oxford University Press, Oxford, 1967.

[Łukasiewicz 53] W. Łukasiewicz, A system of modal logic, *Journal Computing Systems*, vol. 1, p. 111-149, 1953.

[Łukasiewicz 70] W. Łukasiewicz, *Selected Works*, L. Borkowski (éditeur), North-Holland, Amsterdam, 1970.

[McCarthy 80] J. McCarthy, Circumscription : a form of non-monotonic reasoning, *Artificial Intelligence*, vol. 13, no 1-2, p. 27-39, 1980.

[McCarthy 86] J. McCarthy, Applications of circumscription to formalizing common-sense knowledge, *Artificial Intelligence*, vol. 28, p. 89-116, 1986.

[Mendelson 64] E. Mendelson, *Introduction to Mathemetical Logic*, Van Norstrand, Princeton, 1964.

[Meyer and Wieringa 93] J. Meyer and R. Wieringa, *Deontic Logic in Computer Science*, John Wiley, Chichester, 1993.

[Meyer and van der Hoek 95] J. Meyer and W. van der Hoek, *Epistemic Logic for A.I. and Computer Science*, Cambridge University press, Cambridge, 1995.

[Moisil 72] G. Moisil, *Essai sur les Logiques non Chrysippiennes*, Editions de l'Académie de la République Socialiste de Roumanie, Bucarest, 1972.

[Montague 73] R. Montague, The proper treatment of quantification in ordinary English, in Hintikka et al. (éditeurs), *Approaches to Natural Language*, p. 221-242, Reidel, Dordrecht, 1973. Egalement traduction de F. Guenthner, in F. Nef (éditeur), *L'analyse logique des langues naturelles (1968-1978)*, Editions du. C.N.R.S., p. 135-159, 1984.

[Montague 74a] R. Montague, On the nature of certain philosophical entities, *The Monist*, vol. 53, p. 159-194, 1960; repris dans R. Thomason (éditeur), *Formal Philosophy : Selected Papers of Richard Montague*, pp. 148-187, Yale University Press, 1974.

[Montague 74b] R. Montague, Pragmatics, in R. Klibansky (éditeur), *Contemporary Philosophy : A Survey*, p. 102-122, La Nuova Italia Editrice, 1968; repris dans : R. Thomason (éditeur), *Formal Philosophy : Selected Papers of Richard Montague*, p. 95-118, Yale University Press, 1974.

[Montague 74c] R. Montague, English as a formal language, B. Visentini et al. (éditeurs), *Linguaggi nella e nella Tecnica*, p. 189-224, Edizioni di Commutà, 1970; repris dans : R. Thomason (éditeur), *Formal Philosophy : Selected Papers of Richard Montague*, p. 108-221, Yale University Press, 1974.

[Montague 74d] R. Montague, Universal grammars, Theoria. vol. 36, p. 373-398, repris dans : R. Thomason (éditeur), *Formal Philosophy : Selected Papers of Richard Montague*, p. 222-246, Yale University Press, 1974.

[Muskens 88] R. Muskens, Going partial into Montague grammar, ITLI Prepublication series, 1988; également dans R. Bartsch et al. (éditeurs), *Proceedings of the Sixth Amsterdam Colloquium*, Foris Publications, Dordrecht, 1989.

[Mutombo 95] M. Mutombo, Remarques sur l'argument du slingshot, *Revue Philosophique de Kinshasa*, vol. 15-16, p. 27-43, 1995.

[Nef 84] F. Nef (éditeur), *L'analyse Logique des Langues Naturelles*, Editions du Centre National de la Recherche Scientifique, Paris 1984.

[Nef 91] F. Nef, *Logique, Langage et Réalité*, Editions Universitaires, Paris, 1991.

[Nef 98] F. Nef, *L'objet Quelconque. Recherches sur l'Ontologie de l'Objet*, Vrin, Paris, 1998.

[Nerode and Shore 93] A. Nerode and R. Shore, *Logic for Applications*, Springer-Verlag, Berlin, 1993.

[Ntambue 98] R. Ntambue, *Eléments de Logique Trivalente. Du Calcul Logique Trivalent à sa Modalité Ontique*, Bruylant-Academia, Louvain-la-Neuve, 1998.

[Øhrstrøm and Hasle 95] P. Øhrstrøm and P. Hasle, *Temporal Logic. From Ancient Ideas to Artificial Intelligence*, Kluwer Academic Press, 1995.

[Parikh 78] R. Parikh, The completeness of propositional dynamic logic, in *Lecture Notes in Computer Science*, vol. 64, p. 403-415, Springer-Verlag, Berlin, 1978.

[Pereira and Warren 80] F. Pereira and D. Warren, Definite clause grammar for language analysis - A survey of the formalism and a comparison with ATN, *Artificial Intelligence*, vol. 13, p. 231-278, 1980.

[Popkorn 94] S. Popkorn, *First Steps in Modal Logic*, Cambridge University Press, Cambridge, 1994.

[Priest and Sylvan 92] G. Priest and R. Sylvan, Simplified semantics for basic relevant logic, *Journal of Philosophical Logic*, vol. 21, p. 217-232, 1992.

[Prior 57] A. Prior, *Time and Modality*, Oxford University Press, Oxford 1957.

[Prior 62] A. Prior, *Formal Logic*, Clarendon Press, 1962.

[Prior 67] A. Prior, *Past, Present and Future*, Clarendon Press, Oxford 1967.

[Quine 40, 81] W. Quine, *Mathematical Logic*, Revised Edition, Harvard University Press, Harvard 1981.

[Quine 72a] W. Quine, *Logique élémentaire*, traduction de J. Largeault et B. Saint-Sernin, Colin, Paris, 1972.

[Quine 72b] W. Quine, Methodological reflections on current linguistic theory, in D. Davidson and G. Harman (éditeurs), *Semantics of Natural Language*, p. 442, 454, Reidel, Dordrecht, 1972.

[Quine 73] W. Quine, *Méthodes de Logique*, traduction M. Clavelin, 3e édition, Armand Colin, Paris, 1973.

[Quine 76, 81] W. Quine, Worlds Away, in *Journal of Philosophy*, 1976; repr. in *Theories of Things*, The Belknap Press, Harvard, p. 124-128, 1981.

[Quine 93] W. Quine, *La Poursuite de la Vérité*, traduction de M. Clavelin, Editions du Seuil, Paris, 1993.

[Ramsay 88] A. Ramsay, *Formal Methods in Artificial Intelligence*, Cambridge University Press, 1988.

[Rautenberg 79] W. Rautenberg, *Klassische und Nichtklassische Aussagenlogic*, Vieweg, Braunschweig, 1979.

[Rautenberg 83] W. Rautenberg, Modal tableau calculi, *Journal of Philosophical Logic*, p. 403-423, 1983.

[Read 93] S. Read, The slingshot argument, in *Logique et Analyse*, p. 195-218, 1993.

[Rescher 69] N. Rescher, *Many-Valued Logic*, McGraw Hill, New York, 1969.

[Rescher and Urquhart 71] N. Rescher and A. Urquhart, *Temporal Logic*, Springer-Verlag, Berlin 1971.

[Reyes 94] M. Reyes, Referential structure of fictional texts, in J. MacNamara and G. Reys (éditeurs), *The Logical Foundations of Cognition*, p. 309-324, Oxford University Press, Oxford, 1994.

[Riche 91] J. Riche, *Decidability, Complexity and Automated Reasoning in Relevant Logic*, Dissertation doctorale, Australian National University, Dept. of Philosophy, ARP, RSSS 1991.

[Rivenc et de Rouilhan 97] F. Rivenc et de Rouilhan, Traduction et introduction à R. Carnap; voir R. Carnap, *Signification et nécessité*, 1997.

[Sahlqvist 75] H. Sahlqvist, Completeness and correspondence in the first and second order semantics for modal logic, in S. Kanger (éditeur), *Proceedings of the Third Scandinavian Logic Symposium*, pp. 110-143, North-Holland Publishing Company, Amsterdam, 1975.

[Shoam 95] Y. Shoam, *Artificial Intelligence Techniques in Prolog*, 1995.

[Sifakis 82] J. Sifakis, A unified approach for studying the properties of transition systems, *Theoret. Comput. Sci.*, vol. 18, p. 227-259, 1982.

[Smullyan 68] R. Smullyan, *First–Order Logic*, Springer-Verlag, Berlin, 1968.

[Stalnaker 68] R. Stalnaker, A theory of conditionals, in N. Rescher (éditeur), *Studies in Logical Theory, American Philosophical Quarterly*, Blackwell, Oxford, 1968.

[Stenning 76] N. Stenning, A data transfer protocol, *Computer Networks*, vol. 1, p. 99-110, 1976.

[Sterling et Shapiro 94] L. Sterling et E. Shapiro *The Art of Prolog*, MIT Press, Cambridge, Mass., 1994.

[Tarski 44] A. Tarski, The semantic conception of truth, *Philosophy and Phenomenological Research*, vol. 4, p. 341-375, 1944; repris dans : L. Linsky (éditeur), *Semantics and the Philosophy of Language*, p. 13-47, University of Illinois Press, Urbana, 1972. Repris aussi dans A. Tarski, *Logique, Sémantique, Métamathématique, 1923–1944, tome 2*, traduit sous la direction de G. Granger, Armand Colin, p. 265-305, Paris, 1974.

[Thomason 70] R. Thomason, Some completeness results for modal predicate calculi, dans Lambert, 1970.

[Thayse et al. 88] A. Thayse, *Approche Logique de l'Intelligence Artificielle, tome 1 : De la Logique Classique à la Programmation Logique*, Dunod, Paris, 1988.

[Thayse et al. 89] A. Thayse, *Approche Logique de l'Intelligence Artificielle, tome 2 : De la Logique Modale à la Logique des Bases de Données*, Dunod, Paris, 1989.

[Thayse et al. 90] A. Thayse, *Approche Logique de l'Intelligence Artificielle, tome 3 : Du Traitement de la Langue à la Logique des Systèmes Experts*, Dunod, Paris, 1990.

[Thayse et al. 91] A. Thayse, *Approche Logique de l'Intelligence Artificielle, tome 4 : De l'Apprentissage Artificiel aux Frontières de l'IA*, Dunod, Paris, 1991.

[Turner 84] R. Turner, *Logics for Artificial Intelligence*, Ellis Horwood Limited, Chichester 1984. Traduit par P. Besnard, *Logiques pour l'Intelligence Artificielle*, Masson, Paris.

[Ullman 86] J. Ullman, Implementation of logical query languages for databases, *ACM Transactions on Database Systems*, vol. 10, no 3, p. 289–321, 1986.

[Ullman 88] J. Ullman, *Principles of Databases and Knowledge-Base Systems*, Computer Science Press, 1988.

[van Benthem 76] J. van Benthem, *Modal correspondence Theory*, Doctoral Dissertation, Mathematical Institute, University of Amsterdam, 1976.

[van Benthem 91] J. van Benthem, *Language in Action, Categories, Lambdas and Dynamic Logic*, North-Holland, Amsterdam, 1991.

[van Benthem 95] J. van Benthem, Temporal logic; in D. Gabbay, C. Hogger and J. Robinson (éditeurs), *Handbook of Logic in Artificial Intelligence and Logic Programming; vol. 4 : Epistemic and Temporal Reasoning*, Oxford Science Publications, 1995.

[van Benthem 96] J. van Benthem, *Exploring Logical Dynamics*, CSLI Publications and FoLLI, Stanford, 1996.

[van Heijenoort 67] J. van Heijenoort, *From Frege to Gödel. A Source Book in Mathematical Logic*, p. 1879-1931, Harvard University Press, Cambridge Mass., 1967.

[von Wright 51, 57] von Wright, Deontic Logic, *Mind* 1951, repris dans *Logical Studies*, p. 58, 74, Routledge and Kegan Paul, Londres 1957.

[Vuillemin 84] J. Vuillemin, *Nécessité ou contingence. L'Aporie de Diodore et les Systèmes Philosophiques*, Editions de Minuit, Paris 1984.

[Winston 92] P. Winston, *Artificial Intelligence*, Addison-Wesley, Reading (Mass.), 1992 (3e édition).

[Wolper 83] P. Wolper, Temporal logic can be more expressive, *Information and Control* vol. 56, p. 72-99, 1983.

[Wolper 85] P. Wolper, The tableau method for temporal logic : an overview, *Logique et Analyse*, p. 119-136, 1985.

Index

A
Abstrait
de prédicat, 239
Accessibilité
relation d'—, 36, 45
Action
contrôlée, 250
Adéquate
logique —, 50
Adéquation, 152
faible, 162
forte, 162
Aléthique
logique —, 44, 64, 117
Algorithme
d'Euclide, 344
de Peterson, 351
Antimonotonie, 228
Apprentissage, 271, 329
Arborescent
temps —, 25, 127
Arbre, 265, 267–269, 271
de recherche, 281
Arité, 274
Artificielle
intelligence —, 249
Assertion, 344
Atome, 274
Atomique
terme —, 274
Automate, 296
à pile, 297
fini, 296

B
Barcan
formule de —, 112, 125, 223, 224
Base
de connaissance, 278

C
Canonique
modèle —, 155, 288
structure —, 154
système —, 155
valuation —, 155
Chaîne
polarisée, 370
Classe
validité dans une —, 134, 138
Clause, 276
Commune

connaissance —, 96
Complète
logique —, 50
Complexité, 250
Complétude, 152, 205, 209, 232
preuve de —, 152, 156
Comportement
intelligent, 250
Composé
terme —, 274
Composition, 299
Concept, 108, 118
Conceptuelle
interprétation —, 118
Condition
de vérification, 366
Connaissance, 273
commune, 96
distribuée, 96
représentation de la —, 273
base de —, 278
Connectivité
polarisée, 371
Constant
domaine —, 226
Conséquence
globale, 133, 140, 143, 147
locale, 133, 140, 142, 144, 147
logique, 133, 139
Contexte
intensionnel, 243
Contrôlée
action —, 250
Contrôle
point de —, 345
Correction, 152, 205, 206, 232
Correspondance
théorie de la —, 170
Coupure
règle de —, 200

Déclarative
sémantique —, 280
Déontique
logique —, 26, 62, 65
Dérivation, 287
Désignateur
non rigide, 107, 116
Désignateur
rigide, 107
Déterminée
logique —, 51
Degré
d'une formule, 201
modal, 201
Distribuée
connaissance —, 96
Domaine
constant, 226
fixe, 109, 111, 118, 226
variable, 36, 109, 113, 115, 119, 223, 226
Dynamique
logique —, 60, 65
Décidabilité, 169, 205
Déduction
naturelle, 186
théorème de la —, 148
Désignateur
rigide, 241

E

Echecs
jeu d'—, 252
Ecole
mégarique, 23, 62, 123
stoïque, 61, 123
Elimination, 370
Enigme, 257, 306
Entrée
résolution d' —, 288
Epistémique
logique —, 92, 128
Espace
d'état, 343
Etat
espace d'—, 343

Euclide
algorithme d' —, 344
Exécution
de programme, 345
Exclusion
mutuelle, 352
Explicites
tableaux —, 191
Extension, 107

F
Faible
adéquation —, 162
Fait, 276
Filtration, 165, 166
relation de —, 166
Fini
automate —, 296
Fixe
domaine —, 109, 111, 118, 226
Foncteur, 274
Fonction
récursive, 299
Fondamental
terme —, 274
Formes
reconnaissance des —, 256
Formule
de Barcan, 112, 125, 223, 224
pertinente, 370
degré d'une —, 201
Forte
adéquation —, 162

G
Globale
conséquence —, 133, 140, 143, 147
Graphe
de modèle, 210

H
Hanoï,
tours de —, 262
Hexapion
jeu d'—, 268, 329
Hoare
triplet de —, 344

I
Implication
relevante, 38
stricte, 30, 38
Implicites
tableaux —, 198
Instance, 275
Instruction, 343
Intelligence, 249
artificielle, 249
Intelligent
comportement —, 250
programme —, 258
Intension, 107, 118, 243
Intensionnel
contexte —, 243
Interpolant, 369
polarisé, 369
Interpolation, 369
Interprétation
conceptuelle, 118
substantielle, 119
Invariance
propriété d'—, 344
Invariant, 344

J
Jeu
d'échecs, 252
d'Hexapion, 268, 329
de Nim, 267

L
Leibniz
loi de —, 127
Lewis
systèmes de —, 31, 74
Libre
logique —, 114
Linéaire
Logique —, 84
Liste, 275
vide, 275

Locale
conséquence —, 133, 140, 142, 144, 147
Logique, 50
épistémique, 128
adéquate, 50
aléthique, 44, 64, 117
complète, 50
déontique, 26, 62, 65
déterminée, 51
dynamique, 60, 65
libre, 114
multimodale, 29, 59
multivaluée, 66
normale, 51, 125
temporelle, 121, 218
trivalente, 66
épistémique, 92
conséquence —, 133, 139
linéaire, 84
nécessité —, 22
normale, 83
programmation —, 271, 273
temporelle, 81
Loi
de Leibniz, 127

M
Mégarique
école —, 23, 62, 123
Mémoire, 344
Machine
de Turing, 298
Minimal
modèle —, 288
Minimisation, 299
Modèle, 45
canonique, 155, 288
graphe de —, 210
minimal, 288
valide dans un —, 46, 134
Modal
degré —, 201
Modalité, 56
Modus Ponens, 86
Monde
possible, 34
Monotonie, 228
Multimodale,
logique —, 29, 59
Multivaluée
logique —, 66
Mutuelle
exclusion —, 352

N
Nécessité
logique, 22
ontique, 22
règle de —, 51
Négative
polarité —, 368
Naturelle
déduction —, 186
Neuronal
réseau ¬, 254
Nim
jeu de —, 267
Normale
logique —, 51, 83, 125

O
Objet, 108, 273
Ontique
nécessité —, 22
Opérationnelle
sémantique —, 280

P
Paire, 275
Palindrome, 297
Pertinente
formule —, 370
Peterson
algorithme de —, 351
Pile
automate à —, 297
Place
prédicat de —, 345
Point

de contrôle, 345
Polarisé
interpolant —, 369
Polarisée
chaîne —, 370
connectivité —, 371
Polarité, 368
négative, 368
positive, 368
Positive
polarité —, 368
Possible
monde —, 34
Prédicat
de place, 345
Preuve
de complétude, 152, 156
Primitive
récursion —, 299
Processus, 344
Programmation
logique, 271, 273
Programme, 258, 343
intelligent, 258
exécution de —, 345
Prolog, 273
Propriété, 344
d'invariance, 344
de sûreté, 344
Prédicat
abstrait de —, 239

Q
Quantificateur
restreint, 122
Question, 278

R
Récursion
primitive, 299
Récursive
fonction —, 299
Réduit
type —, 368
Réfutation, 287
Réseau
neuronal, 254
Résolution
d'entrée, 288
unitaire. 286
Règle, 276
de nécessité, 51
Ramifié
temps —, 25, 127
Rang, 368
Recherche
arbre de —, 281
Reconnaissance
des formes, 256
Relation, 276
d'accessibilité, 36, 45
de filtration, 166
Relevante
implication —, 38
Représentation
de la connaissance, 273
Reste, 275
Restreint
quantificateur —, 122
Rigide
désignateur —, 107, 241
désignateur non —, 116
terme —, 111
Règle
de coupure, 200

S
Sémantique, 280
déclarative, 280
opérationnelle, 280
Sûreté
propriété de —, 344
Simulation, 252
Stoïque
école —, 61, 123
Stricte
implication —, 30, 38
Structure, 45
canonique, 154

valide dans une —, 46
validité dans une —, 135
Substance, 108, 119
Substantielle
interprétation —, 119
Substitution, 136
Systèmes
de Lewis, 31, 74
Système
canonique, 155

Tête, 275
Tableaux
explicites, 191
implicites, 198
Tactique, 263, 265, 267, 268
Tautologie, 50
Temporelle
logique —, 81, 121, 218
Temps
arborescent, 25, 127
ramifié, 25, 127
Terme, 273
atomique, 274
composé, 274
fondamental, 274
rigide —, 111
Théorème, 50
de la déduction, 148
Théorie
de la correspondance, 170
Tours
de Hanoï, 262
Transition, 343
Triplet
de Hoare, 344
Trivalente
logique —, 66
Turing
machine de —, 298
Type, 367
réduit, 368

U

Ultrafiltre, 181
Unitaire
résolution —, 286

V

Vérification
condition de —, 366
Valide, 46, 134
dans un modèle, 46, 134
dans une structure, 46, 135
dans une classe, 134, 138
Valuation, 136
canonique, 155
Variable, 274, 343
domaine —, 36, 109, 113, 115, 119, 223, 226
Vide
liste —, 275